THOMSON
COMPONENTS
MOSTEK

COMMUNICATIONS PRODUCTS

APPLICATION NOTE

MK68590 (LANCE™) INTERFACE TO MK68000

TABLE OF CONTENTS

PARAGRAPH TITLE PAGE

LIST OF ILLUSTRATIONS

FIGURE NUMBER — TITLE — PAGE

1.0 INTRODUCTION

The LANCE™ Application Note describes the basic hardware and software needed to interface the MK68590, Local Area Network Controller for Ethernet (LANCE), to the MK68000, Mostek's 16/32 microprocessor. It can be used as a "cookbook" for designing the basic hardware and software needed for a LANCE-68000 interface, but is actually more useful as a design guide.

This publication contains seven sections in addition to this introduction. Section two is the Ethernet Primer, which contains background material and basic concepts involved in Ethernet communication. Section three defines terms used throughout the Application Note. Section four is an overview of the Local Area Network Controller for Ethernet (LANCE) chip. Section five is the Ethernet Node overview, listing design requirements for an intelligent Ethernet node. It deals with the block level concepts of both hardware and software. Section six is the Hardware Description. It deals with the basic hardware requirements and includes both block diagrams and schematics. Section seven describes the software used in the LANCE-MK68000 interface. This section includes flowcharts, assembly code, and a step-by-step explanation of the software. The eighth and final section is a Assembly Code listing of the software needed for an intelligent Ethernet node.

The LANCE-MK68000 hardware and software were designed as simply as possible. The design incorporates a minimum number of interface logic gates and no time multiplexing of devices. Structured assembly code with few subroutines and loops is used, making it easy to change and understand. The schematics and assembly code are presented in a format that easily lends itself to a self-tutorial on an existing Ethernet interface design for users. The design is intended to give users ideas on how to design or how to improve present designs. But, if so desired, users can apply the information directly to quickly generate a basic working interface.

This Application Note refers to many technical aspects of the LANCE but they are not fully explained. For this reason, users should also study the MK68590 LANCE Technical Manual, published by United Technologies Mostek.

LANCE is a trademark of Mostek Corporation.

2.0 ETHERNET PRIMER

Much debate and questioning as to the nature of celestial bodies existed in the early days of science. These questions included: What type of material exists between the heavenly bodies? If it is not a vacuum, how does light move across it? The conclusion agreed upon was that an Ether existed between the planets and it was the medium light used to travel from the sun to the Earth.

Xerox borrowed this terminology in their early stages of defining a local area network. Ether was the medium information would use to travel from one station to the next throughout the network specification. They called this network "Ethernet."

Ethernet was designed as a system for local communication between computing stations. A series of tapped coaxial cables between computing stations comprise a network that connects up to 1024 different stations per network. In addition, a gateway interface may be used to connect each network to other networks. The computing stations may consist of personal computers, CAD workstations, file storage devices, magnetic tape backup stations, large central computers, printers, plotters or any device that conforms to the Ethernet standard.

The objective of Ethernet was to provide a communication system that can grow with users' needs and accommodate several buildings in a local area. One of its purposes is to eliminate bottlenecks and reliability problems associated with a central controller.

Ethernet provides for one, and only one, connection between any two points on the network. Since the Ether, or the common broadcast communication channel, is passive when an active node fails, only its operation is affected - not the entire system.

Ethernet employs Carrier Sense Multiple Access with Collision Detect (CSMA/CD). The principle behind this operation is similar to human conversation. If several people are conversing while standing in a circle (assuming it is a non-controversial discussion between polite adults), only one person speaks at a time. When the first person finishes, someone else begins to speak and continues until he or she is also finished. If no one has anything else to say, silence falls over the group.

The same is true for Ethernet. Each node listens until the network becomes silent. Then, if a message needs to be transmitted the node broadcasts onto the Ether and the remaining nodes listen.

The more people in the circle, and the more they have to say, the greater the chance that two or more people will begin speaking simultaneously. If this happens in normal conversation, all the participants who have started speaking will stop and whoever begins to speak first will be able to say what he has to say. Two or more people may occasionally begin speaking again at the same time, in which case, they will again stop talking and, hopefully, the more gracious member of the group will wait. When a person starts talking, no one else begins until that person finishes. Delays and collisions occur in Ethernet just as they do in normal conversation. Due to the propagation delays encountered in the medium, a station may not sense network activity and begin transmitting. Once it begins, it keeps listening to the network; if it detects a collision, it aborts the transmission and waits a certain amount of time determined by a random number generator.

In any given group of people, some are naturally talkers while others are listeners. If the conversation does not concern them, the listeners are less likely to pay attention. The same is true with Ethernet stations. Some stations, such as personal computers, transmit much information and others, such as printers, do much of the receiving.

A particular station's Ethernet interface connects bit-serially through an interface to a transceiver that taps into the passive Ether. The Ethernet station, or node, broadcasts its message into the passive Ether, enabling all nodes on the network to hear it. All stations receive the message and determine if they are the desired destination. If the transmitting node's destination address matches that of the receiving node, the packet is accepted and the station digests the data. If the address does not match, the receiving node rejects the packet.

A station may use two addressing schemes when broadcasting. The first is the physical addressing scheme, whereby the transmitting station addresses one, and only one unique destination station. As long as all stations on the network are not in the promiscuous mode, only one station accepts the message.

The second adddressing scheme uses a logical address. In this operation, stations receiving the message must determine if they are one, of possibly many, intended recipients. An example of logical addressing is one in which all printers have the same logical address. If people want to send memos via the printers, they simply set the destination as the logical address — "printers." All printers on the network then see themselves as the destination and all accept the message. In addition, a station may set itself up in a promiscuous mode. In this mode, the station accepts all incoming messages, no matter what destination address the message has.

When Xerox defined Ethernet, their intent was to define a standard that all manufacturers who wanted access to the Ether, could use. They wanted to define a rigorous standard from the onset to avoid incompatability. Universal acceptance of a standard is the very key to practical applications of local area networking. The International Standards Organization (ISO) has approved a layered protocol standard that specifies functions, as well as minimal rules for accessing these functions and for information exchange between devices on the network. This seven-layer architecture logically groups functions and provides conventions for connecting functions between layers. The model, shown in Figure 1, is called the Open Systems Interconnection (OSI) network model.

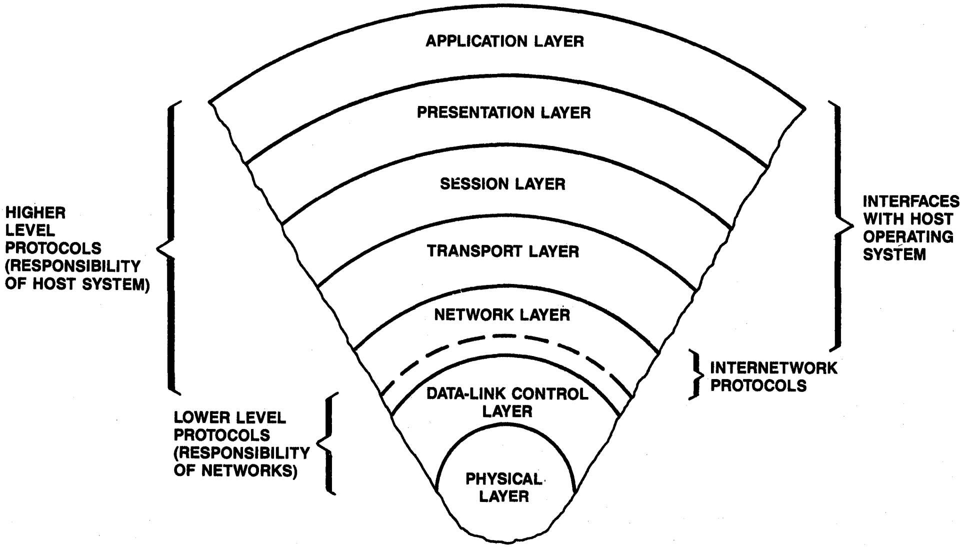

APPLICATION LAYER
PRESENTATION LAYER
SESSION LAYER
TRANSPORT LAYER
NETWORK LAYER
DATA-LINK CONTROL LAYER
PHYSICAL LAYER
HIGHER LEVEL PROTOCOLS (RESPONSIBILITY OF HOST SYSTEM)
LOWER LEVEL PROTOCOLS (RESPONSIBILITY OF NETWORKS)
INTERFACES WITH HOST OPERATING SYSTEM
INTERNETWORK PROTOCOLS

Figure 1. OSI Network Model

The bottom three layers of the OSI model include the physical, data-link control, and network layers. Hardware is based on the actual definition of the two lower layers. Specifications in these layers include the transmission medium (Ether) and how the node must interface to the Ether. The physical and data-link control layers also specify how information should be formatted for error-free transmission and reception. Each layer supports another in hierarchial fashion. In other words, layer 1 serves layer 2, layer 2 serves layer 3 and so forth. The three bottom layers differ according to network architecture. The top three layers — session, presentation, and application — are the same for all networks. The transport layer is the interfacing layer between the top three and bottom three layers.

The functions of each layer of the OSI model follow.

Physical Layer:

- Handle cables, connectors, and components
- Handle collision detection for CSMA/CD
- Handle voltages and electrical pulses

Data Link Control Layer:

- Make sure data is not mistaken for flags
- Add error checking algorithms
- Insert flags to indicate beginning and end of messages
- Provide access methods for local area networks

Network Layer:

- Internetworking
- Send control messages to peer layers about own status
- Set up routes for packets to travel (virtual circuit)
- May disassemble transport messages into packets and reassemble them at their destination
- Flow control
- Recognize message priorities and send messages in proper priority order
- Address network machines on the route through which the packets travel

Transport Layer:

- Multiplex end-user addresses onto network
- Monitor quality of service
- End-to-end error detection and recovery
- Address end user machines without concern for route of message or address machines in route between end user machines
- Possible disassemble and reassemble session messages
- Map address to names

Session Layer:

- Send information from one task to another
- Coordination and cooperation between end users tasks
- Start and stop tasks
- Dialog control
- Recovery from communication problems during a session without losing data

Presentation Layer:

- Encoding and decoding
- Data compaction
- Syntax transformation for character sets, text string, data display formats, graphics, file organization, data types

Application Layer:

- Log in
- Password checks
- Color control
- Graphics procedures
- Downline loading
- Creation of charts and displays
- File requests and file transfers

- Remote job entry
- Computer based message systems
- Job minipulation
- Virtual terminal service
- Data-based queries, insertions, and deletions
- User specific applications (e.g. editing, word processing, electronic funds transfer, airline reservation, and transaction processing)

The LANCE, together with the SIA, Transceiver, and Coaxial cable satisfy the specifications in the bottom two layers of the ISO model. By adding the software shown in this Application Note, layers 1, 2, and 3a can be satisfied.

3.0 DEFINITION OF TERMS

DMA Capability: The ability to directly access memory or memory-mapped I/O. This includes reading, writing, and control signal generation.

Peripheral: A processing unit attached to the system bus which handles part of the processing load.

Bus Master: A CPU or DMA device that has gained control of the system bus. It can initiate data transfers on the bus by issuing an address and by driving the read/write, address strobe, and data strobe control signals.

Bus Slave: A device that decodes the address, read/write, address strobe, and data strobe control signals and responds accordingly for a read or write operation.

Bus Arbitration: In a system with more than one device capable of being the Bus Master, a bus arbitration convention must be employed to determine which device may take control of the system bus at any one time. Normally one of the Bus Master type devices is responsible for receiving and granting requests for access to the system bus.

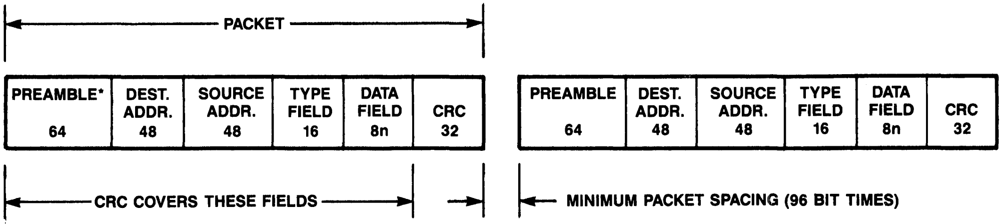
Front End Processor: A processor microsystem, usually consisting of a microprocessor, or single-chip microcomputer, along with memory and control logic. This microsystem alleviates some of the burden placed on the main host processor. The front end processor acts as an intermediate stage of processing between the host processor and an I/O device.

Intelligent Ethernet Node: An Ethernet node that acts as a front end processor to the host processor. An intelligent Ethernet node would basically consist of microprocessor or microcomputer, memory, an Ethernet protocol device (LANCE), transceiver interface device (SIA), and the associated firmware required to implement the lower layers of the Ethernet protocol.

4.0 LANCE CHIP DESCRIPTION

The MK68590 LANCE (Local Area Network Controller for Ethernet) is a 48-pin VLSI device that simplifies interfacing a microcomputer or minicomputer to an Ethernet Local Area Network (LAN). This chip operates in a local environment that includes a closely coupled memory and microprocessor. The LANCE uses scaled N-channel MOS technology and is compatible with several popular microprocessors. It interfaces to a microprocessor bus characterized by time-multiplexed address and data lines. Typically, data transfers are 16 bits wide, but byte transfers occur if the buffer memory address boundaries are odd. The address bus is 24 bits wide.

The Ethernet packet format consists of a 64-bit preamble, a 48-bit destination address, a 48-bit source address, a 16-bit type field, and a 46- to 1500-byte data field terminated with a 32-bit CRC (cyclic redundancy check) as shown in Figures 2 and 3. The packet's variable widths accommodate both short status, command and terminal traffic packets, and long data packets to printers and disks (1024 byte disk sectors, for example). Packets are spaced a minimum of 9.6 usec apart to allow one node time enough to receive back-to-back packets.



PREAMBLE*	DEST. ADDR.	SOURCE ADDR.	TYPE FIELD	DATA FIELD	CRC
64	48	48	16	8n	32

PREAMBLE	DEST. ADDR.	SOURCE ADDR.	TYPE FIELD	DATA FIELD	CRC
64	48	48	16	8n	32

*LAST BYTE IS START OF FRAME SYNCHRONIZATION BYTE--10101011

Figure 2. Ethernet and LANCE Packet Format

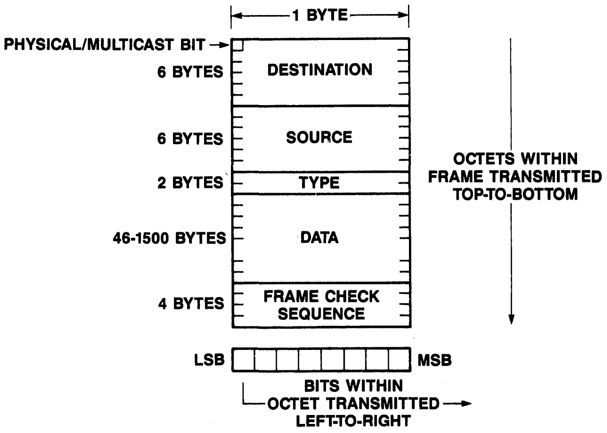


Figure 3. Ethernet and Packet Bit Transmission Sequence

The LANCE operates in a minimal configuration that requires close coupling between local memory and a processor. The local memory provides packet buffering and is a communication link between the chip and processor. During initialization, the control processor loads the starting address of the initialization block plus the operation mode into the LANCE via two control registers. The host processor talks directly to the LANCE only during this initialization as a Bus Slave peripheral. The LANCE's DMA machine, under microword control, handles all further communications.

The LANCE on-chip DMA channel provides flexibility and speed by communicating with the host, or dedicated microprocessor, through common memory locations. Buffer management is organized by a circular queue of tasks in memory called "descriptor rings" (see Figure 4). Separate descriptor rings describe transmit and receive operations. Up to 128 tasks may be queued on a descriptor ring for future execution. Each entry in a descriptor ring holds a pointer to a data memory buffer and an entry for the data buffer length. Data buffers can be chained or cascaded to handle a long packet in multiple data buffer areas. The LANCE searches the descriptor rings in a "look-ahead manner" to determine the next empty buffer for chaining buffers together or handling back-to-back packets. As each buffer is filled, an "own" bit is reset, signaling the host processor to empty this buffer.

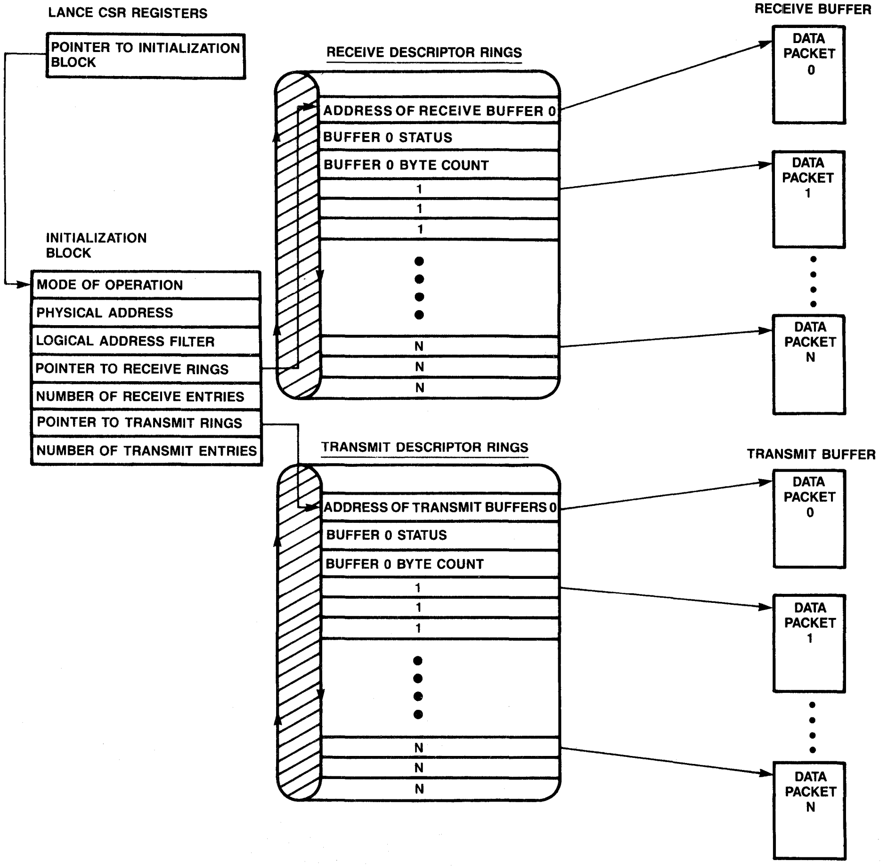


Figure 4. LANCE Memory Management

5.0 ETHERNET NODE OVERVIEW

5.1 INTRODUCTION

The LANCE has two basic types of micro-interfaces. The first is when the LANCE acts as a peripheral to a host processor and the second is when the LANCE and a dedicated microprocessor or microcontroller work together to form an intelligent node. In the second type, all processing between the processor and the LANCE occurs on a local bus, while the entire node interfaces to the main system bus as an intelligent subsystem.

The LANCE parallel interface is designed to be an easy or "friendly" interface to several popular 16-bit microprocessors. This Application Note addresses MK68000 interface requirements, but the concepts can be applied to other microprocessors.

5.2 OPERATIONAL DESCRIPTION

The following two sections are operational descriptions of a typical intelligent Ethernet node controller design, and the Ethernet node controller design used to generate this Application Note.

5.2.1 TYPICAL ETHERNET NODE

An intelligent Ethernet node's primary function is to unburden the host processor from many networking tasks and, at the same time, reduce system bus congestion by eliminating the need for the LANCE to ever access the system bus.

A typical intelligent node performs node initialization and self tests, as well as transmitting and receiving messages. It also retains statistical records of error-causing conditions and generates time-out interrupts. These timeout interrupts are used for both memory refresh and upper-level protocol requirements.

The software of the intelligent node isolates the higher level user software from node details, such as memory refresh and the LANCE interface.

In a typical system, an intelligent Ethernet controller may be designed and placed on a separate system board. The LANCE chip permits placement of the entire Ethernet controller on a single system board because it reduces the chip count. This board would interface to a particular bus structure, i.e., the VME, Versabus™, Multibus™, etc.

The system host procesor may interface to the node processor via a dual-ported memory with the use of semiphores.

5.2.2 APPLICATION-NOTE ETHERNET NODE

The Ethernet node controller design discussed in this Application Note does not have the characteristics of a typical intelligent Ethernet controller. It was designed more as a demonstration and teaching tool. The only software written for the node is the lower level software included in the Appendix.

This design uses a total of 8K bytes of memory. A greater quantity of memory would be present in a typical intelligent Ethernet node design. This design has no memory refresh requirements since static RAMs are used instead of dynamic RAMs.

This design includes a "message send request" button that users depress to send data messages to any desired node. The push button simulates a message request that the system host processor would normally generate. The button idea is only used on the breadboard for controlled message transfer for debug and demonstration purposes.

Depressing the button interrupts the local processor and calls a message routine. This message routine generates a pseudo message and updates the transmit descriptor so proper transmission can occur. Immediately after pushing the button, users can examine the memory to verify message transmission and proper hardware and software operation. The pushbutton was felt to be the best way to demonstrate message generation.

Aside from these few differences, the Ethernet node design used in this publication is functionally the same as a typical design used in industry.

Versabus is a trademark of Motorola, Inc.
Multibus is a trademark of INTEL Corp.

5.3 HARDWARE REQUIREMENTS

As the Bus Master, the LANCE has a wide 24-bit linear address space it can access directly via DMA. This 24-bit address bus interfaces directly to the MK68000's 24-bit bus. The only provision is that latches must be placed on the multiplexed address/data bus to latch up the address at the start of a read or write cycle when the LANCE is the Bus Master. Only the lower 15 bits must be latched up; the upper eight bits can be interfaced directly. When interfacing to the MK68000, BM1 and BM0 correspond to upper data strobe (UDS) and lower data strobe (LDS). Address bit A0 is not used in the basic MK68000 interface.

The basic blocks of this design include the LANCE, MK68000, memory, bus arbitration circuitry, chip select circuitry, and DTACK circuitry. Figure 5 is a block diagram of this interface.

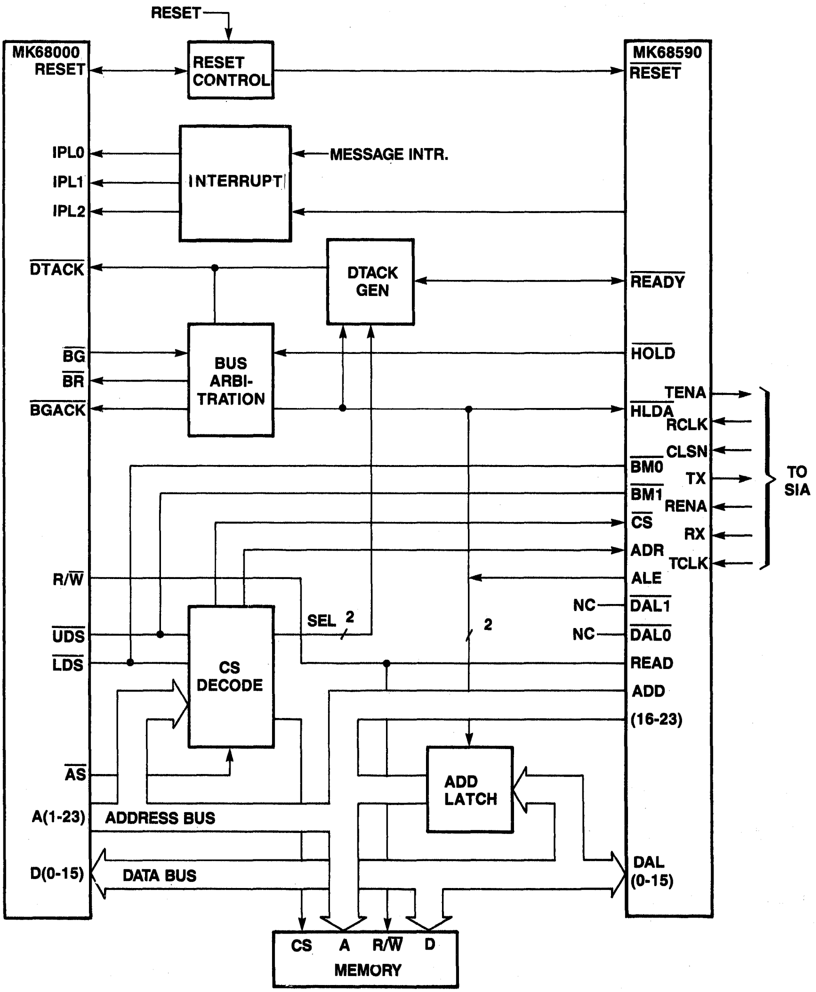


Figure 5. 68000/68590 Block Diagram

The memory includes 8K bytes of random access memory (RAM) and 4K bytes of erasable programmable read only memory (EPROM). The EPROM contains the Ethernet node program, while the RAM is used for receive and transmit rings as well as stack area.

Section 6 — Hardware Description gives a detailed hardware description.

5.4 SOFTWARE REQUIREMENTS

The software program has three functions:

1. Initialize the LANCE

2. Perform message ring management

3. Keep track of the error- and flag-causing conditions.

The software acts as an intermediate stage between the higher-level software protocol of the system host processor and the lower-level protocol implemented by the LANCE. Figure 6 shows the main flow of the software.

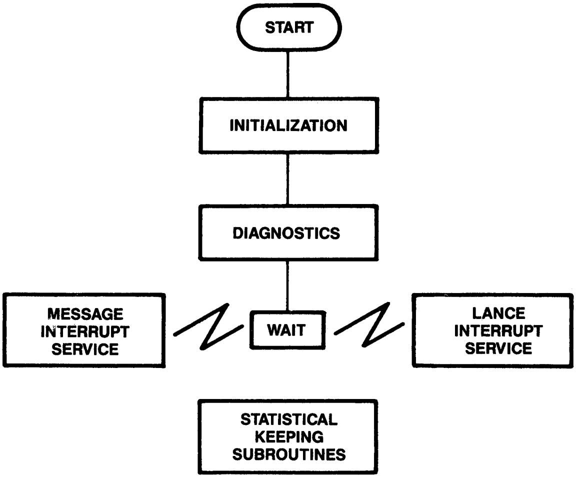


Figure 6. Program Software Overview Flowgraph

The initialization and diagnostics are implemented upon powerup. Users may set up the diagnostics submodule to be implemented any time they desire an operational check. The two service modules — LANCE Interrupt and Message Interrupt — are both interrupt-driven.

The LANCE Interrupt service routine is called when the LANCE interrupts the local processor. The LANCE interrupts the processor when a message is received or transmitted, initialization is complete, or an error has occurred. The error- or flag-causing condition is determined by reading the Control and Status Registers zero (CSR0). The processor then either services the flag-raising condition or calls one of the statistical-keeping subroutines. It then simply reports the condition to the system processor.

It is not necessary for the LANCE to operate on an interrupt-driven basis. The processor may be programmed to periodically poll the Control and Status Registers zero (CSR0) for flag-causing conditions. This Application Note performs the function on an interrupt basis to eliminate the need for a timer.

The message interrupt service routine is called when users depress the "message request" button. The routine generates a pseudo message depending on what parameters users give it. It then writes the message into a transmit message buffer and updates the transmit message descriptor.

As stated before, the "Push-button" operation is not included in a typical Ethernet node processor, but appears here to give the reader possible suggestions on how to write required software for servicing an actual system-generated message.

5.5 SYSTEM MEMORY MAP

All software activities take place in the low end of memory. The entire program uses less than 2K bytes of memory. Figure 7 is a memory map of the space this program uses.

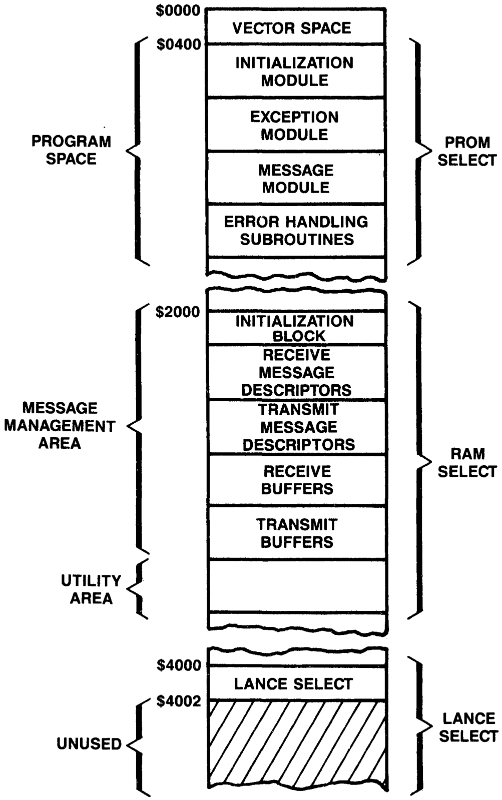


Figure 7. System Memory Map

Memory area from hex address $0000 to $03FF is reserved for vectors. The program resides in the memory space bounded by addresses $0400 and $1FFF. Message management memory space is between $2000 and $387B. This area contains the initialization block and the receive and transmit message descriptors, along with the receive and transmit buffers. The memory space between addresses $387C and $3FFF is the utility area. The program stack, ring management, and error flags reside here. Figure 8 gives a detailed map of the utility area.

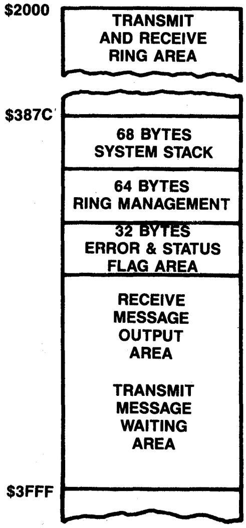

$2000
TRANSMIT AND RECEIVE RING AREA
$387C
68 BYTES SYSTEM STACK
64 BYTES RING MANAGEMENT
32 BYTES ERROR & STATUS FLAG AREA
RECEIVE MESSAGE OUTPUT AREA
TRANSMIT MESSAGE WAITING AREA
$3FFF

Figure 8. Utility Area Memory Map

Memory location $4000 is the address of the LANCE's register data port (RDP). Memory location $4002 is the address of the LANCE's register address port (RAP).

The receive and transmit ring length, as well as their buffer sizes, are all variable. These variables may be altered by changing an equate statement at the beginning of the program. This Application Note uses eight receive descriptors and four transmit buffers. The size for both transmit and receive buffers is 256 bytes. There are twice as many receive buffers as transmit buffers because it is undesirable for the node to miss an incoming packet due to a lack of receive buffers.

5.6 UTILITY AREA DEFINITION

The utility area is composed of the following:

1. 68 bytes of system stack
2. 64 bytes of ring management variables
3. 32 bytes of error and status flag area
4. 620 bytes for message area.

The message storage area is to be used for receive message output and transmit message waiting area. A ring management stack or status area is set aside in memory to record all transmit and receive descriptor ring activity. Figure 9 gives the address location and describes the information in that memory location.

The addressing mode to access all ring management locations is "Address Register Indirect with Displacement". Base address $38C0 is placed in register A4 upon initialization. The displacements are all defined in the equate statements at the beginning of module number one (see Software listing in the Appendix).

The ring management area has several pointers. Some point to the top and bottom of the descriptor rings. Others point to the next-descriptor to be used and the last-descriptor used. The complicated task of message management is controlled by using these pointers and other status information in the ring management area.

ADDRESS DESCRIPTION OF CONTENTS

38C0 RECEIVE RING LENGTH Number of entries in the receive ring

38C2 TRANSMIT RING LENGTH Number of entries in the transmit ring

38C4 RECEIVE RING BASE ADDRESS POINTER This points to the first receive message descriptor

38C8 TRANSMIT RING BASE ADDRESS POINTER This points to the first transmit message descriptor

38CC RECEIVE RING BOTTOM ADDRESS POINTER This points to the last receive message descriptor

38D0 TRANSMIT RING BOTTOM ADDRESS POINTER This points to the last transmit message descriptor

38D4 LAST RECEIVE DESCRIPTOR USED POINTER This points to the last receive descriptor to be used. When the LANCE interrupts the host because of a received message, this pointer will indicate which ring corresponds to the message. If more than one buffer was required for the incoming message, than one buffer was required for the incoming message, this pointer will point to the first descriptor used.

38D8 LAST TRANSMIT DESCRIPTOR USED POINTER This points to the last transmit descriptor turned over to the host by LANCE. When LANCE interrupts the host because it has just transmitted a message, this pointer will indicate the descriptor just used.

38DC NEXT TRANSMIT DESCRIPTOR TO BE USED POINTER This points to the next transmit message descriptor available for use. The "number of rings available" must be checked before the next ring is accessed. If there are "0" rings available the Next Transmit Pointer will be pointing to a ring which has not been serviced by LANCE.

38E0 LOOPBACK MESSAGE COUNT This contains the number of bytes contained in the loopback message. This is not used in diagnostics

38E2 TRANSMIT DESCRIPTOR COUNT number of descriptors that are available

38E4 RING MANAGEMENT STATUS

BIT #1 "1" indicates that LANCE is in LOOPBACK Mode
BIT #2 "1" indicates that the START OF PACKET bit was set before
BIT #3 "1" indicates that the END OF PACKET bit was set before
BIT #4 "1" indicates that the program is in DIAGNOSTIC Mode
BIT #5 "1" indicates that this portion of the TEST IS COMPLETE
BIT #6 NOT USED
BIT #7 NOT USED

38E8 LOOPBACK TRANSMIT BUFFER ADDRESS POINTER

38EC MORE COUNTER keeps count of the number of times more than one retry was needed to transmit a packet

38EE ONE COUNTER keeps count of the number of times exactly one retry was needed to transmit a packet

38F0 MESSAGE WORD used for testing and debugging

Figure 9. Ring Management Area Description

6.0 HARDWARE DESCRIPTION

6.1 INTRODUCTION

The circuit described in this Application Note has a minimal amount of logic for interfacing the MK68000 to the LANCE. Figure 5 is a block diagram of the circuit.

The basic interface blocks consist of the following:
1. Chip select decode and DTACK (Data Transfer Acknowledge) generation
2. Address/Data bus interface
3. Interrupt - autovector
4. Bus arbitration blocks.

Detailed schematics of these blocks are given in Figures 10 through 16.

6.2 CHIP SELECT DECODE CIRCUITRY

The Chip Select Decode circuit consists of five logic gates and a Bipolar Programmable Read Only Memory (PROM) as shown in Figure 10. The circuit output enables both the RAM and EPROM, as well as enabling the LANCE. When the LANCE is the Bus Slave and the MK68000 needs to access one of its control-status registers, the LANCE is enabled. As Bus Masters, either the MK68000 or the LANCE can access memory. The DTACK circuitry uses the output signals, RAM SELECT and ROM SELECT. They generate the needed DTACK and READY signals for the MK68000 and the LANCE, respectively.

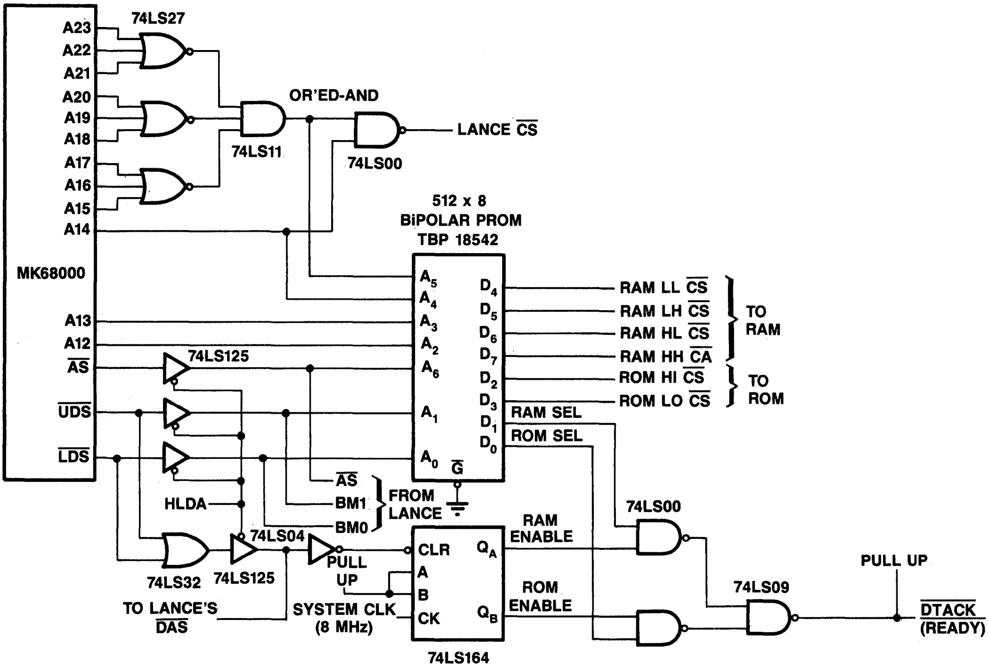
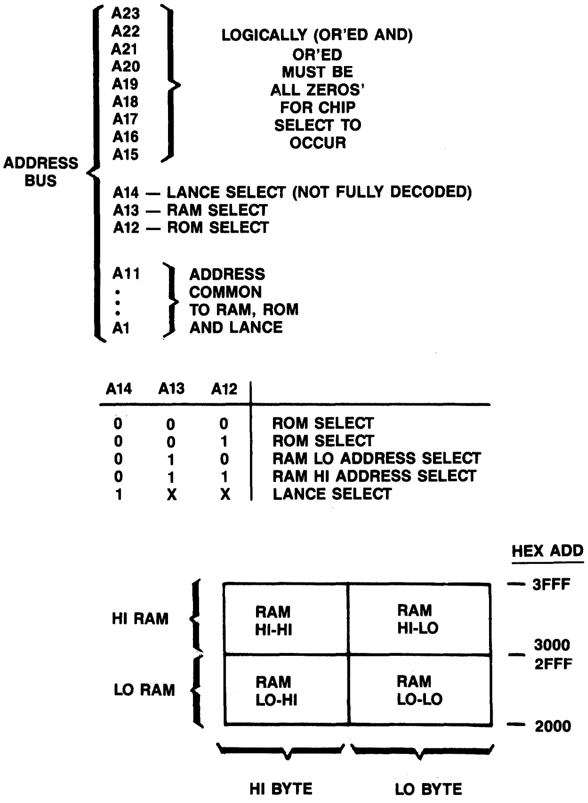


Figure 10. Chip Select Decode and DTACK Circuit

Figure 11 describes chip select decoding. Since the entire addressable memory used in this circuit resides below hex address $7FFF, address bits A15 thru A23 are not used. These address bits are all inputs to the OR-AND gate array. This array's output is active if address bits A15 through A23 are all at logic "0". The decode circuit's output enables different segments of memory. $\overline{\text{RAM HH}}$ enables high-address, high-word of RAM, while $\overline{\text{RAM HL}}$ will enable the high-address, low-word of RAM. $\overline{\text{RAM LH}}$ enables the low-address, high-word of RAM, and so forth. Figure 12 is a chip select decode PROM firmware map.



A14	A13	A12	
0	0	0	ROM SELECT
0	0	1	ROM SELECT
0	1	0	RAM LO ADDRESS SELECT
0	1	1	RAM HI ADDRESS SELECT
1	X	X	LANCE SELECT

Figure 11. Chip Select Decode Description

AS/	ORED AND	A14	A13	A12	UDS/	LDS/		RAM HH CS/	RAM HL CS/	RAM LH CS/	RAM LL CS/	ROM LO CS/	ROM HI CS/	ROM SELECT	RAM SELECT		
A_6	A_5	A_4	A_3	A_2	A_1	A_0	HEX ADD	D_2	D_6	D_5	D_4	D_3	D_2	D_1	D_0	HEX DATA	
0	1	0	0	0	0	0	20	1	1	1	1	0	0	1	0	F2	ROM WORD SELECT
0	1	0	0	0	0	1	21	1	1	1	1	1	0	1	0	F2	ROM HI BYTE SELECT
0	1	0	0	0	1	0	22	1	1	1	1	0	1	1	0	F6	ROM LO BYTE SELECT
0	1	0	0	0	1	1	23	1	1	1	1	1	1	0	0	FC	INVALID (NO SELECT)
0	1	0	0	1	0	0	24	1	1	1	1	0	0	1	0	F2	ROM WORD SELECT
0	1	0	0	1	0	1	25	1	1	1	1	1	0	1	0	FA	ROM HI BYTE SELECT
0	1	0	0	1	1	0	26	1	1	1	1	0	1	1	0	F6	ROM LO BYTE SELECT
0	1	0	0	1	1	1	27	1	1	1	1	1	1	0	0	FC	INVALID (NO SELECT)
0	1	0	1	0	0	0	28	1	1	0	0	1	1	0	1	CD	LOW ADDRESS/WORD SELECT
0	1	0	1	0	0	1	29	1	1	0	1	1	1	0	1	DD	LOW ADDRESS/HI BYTE SELECT
0	1	0	1	0	1	0	2A	1	1	1	0	1	1	0	1	ED	LOW ADDRESS/LO BYTE SELECT
0	1	0	1	0	1	1	2B	1	1	1	1	1	1	0	0	FC	INVALID (NO SELECT)
0	1	0	1	1	0	0	2C	0	0	1	1	1	1	0	1	3D	HI ADDRESS/WORD SELECT
0	1	0	1	1	0	1	2D	0	1	1	1	1	1	0	1	FD	HI ADDRESS/HI BYTE SELECT
0	1	0	1	1	1	0	2E	1	0	1	1	1	1	0	1	BD	HI ADDRESS LO BYTE SELECT
0	1	0	1	1	1	1	2F	1	1	1	1	1	1	0	0	FC	INVALID (NO SELECT)
1	X	X	X	X	X	X		1	1	1	1	1	1	0	1	FC	INVALID (NO SELECT)
X	X	X	X	X	X	X		1	1	1	1	1	1	0	0	FC	INVALID (NO SELECT)

Figure 12. Chip Select Decode PROM Firmware

6.3 DTACK CIRCUITRY

A few logic gates and and a shift register make up the $\overline{\text{DTACK}}$ circuitry (see Figure 10). The shift register allows selectable delay time outputs to compensate for varying memory access speeds.

When the LANCE is a Bus Master, $\overline{\text{READY}}$ is an asynchronous acknowledgement from the $\overline{\text{DTACK}}$ circuit that memory will accept data in a WRITE cycle, or that memory has put data on the Data/Address lines in a read cycle.

As a Bus Slave, the LANCE asserts $\overline{\text{READY}}$ when it has put data on the Data/Address lines during a READ cycle or, is about to take data off the Data/Address lines during a WRITE cycle. Ready is a response to Data Address Strobe ($\overline{\text{DAS}}$) and is negated after $\overline{\text{DAS}}$ is negated. The separate $\overline{\text{DTACK}}$ signals are wire ORed with the $\overline{\text{READY}}$ signal from the LANCE. In this manner, the $\overline{\text{READY}}$ signal can be either an input or output. When the LANCE is the Bus Master, the $\overline{\text{DTACK}}$ input on the MK68000 is in the high impedance state.

6.4 ADDRESS/DATA BUS INTERFACE

Figures 13 and 14 are schematics of address and data bus interfacing. The interfacing is very straightforward. The data bus is connected directly between the MK68000, the LANCE, and memory. Address bits A16 through A23 are connected up directly between the LANCE and the MK68000. Since the LANCE has a multiplexed Address/Data bus, tri-state latches must be placed on the DAL lines. These latches store the address during the beginning of a read or write cycle when the LANCE is a Bus Master. $\overline{\text{HLDA}}$ enables the output of the latches, this occurs only when the LANCE is a Bus Master.

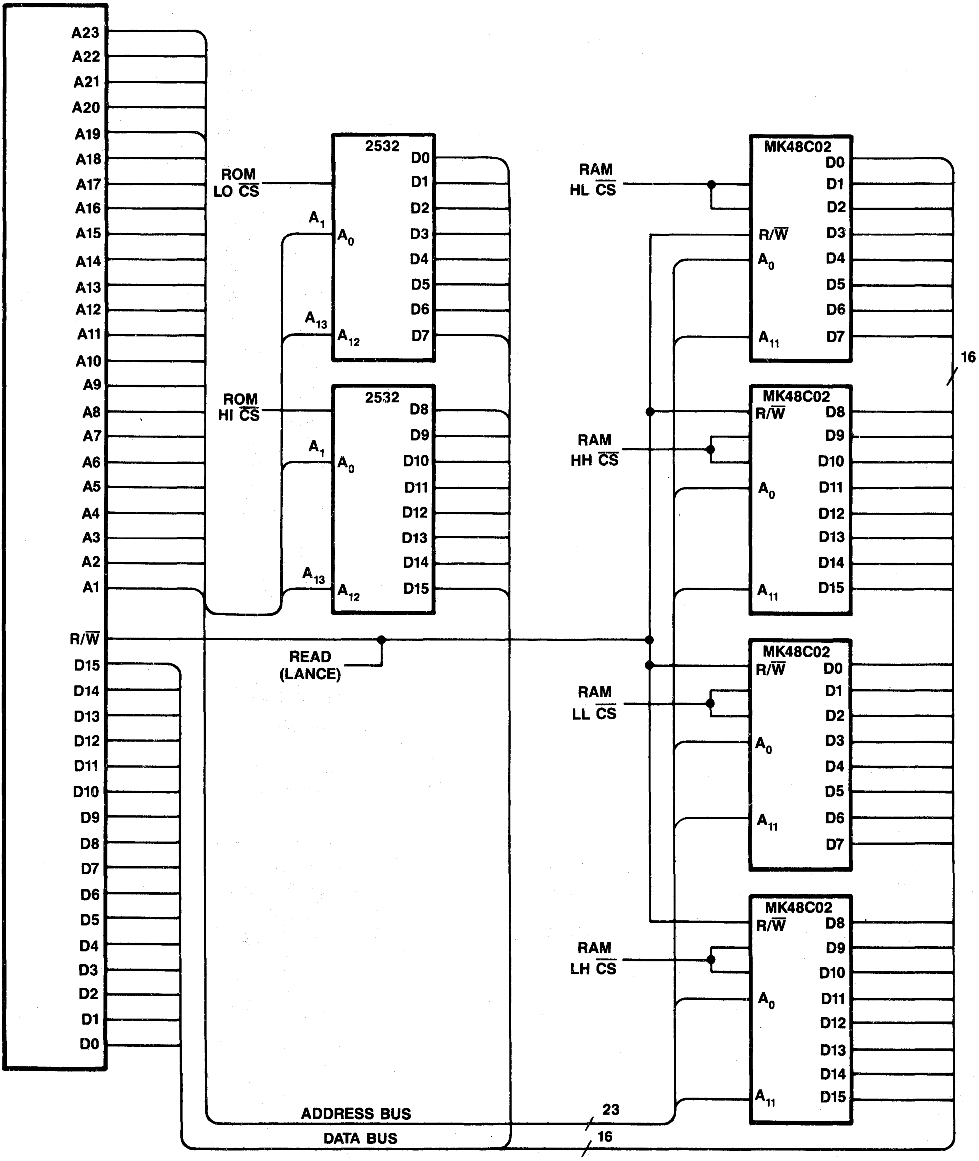


Figure 13. MK68000-Memory Interface Circuit

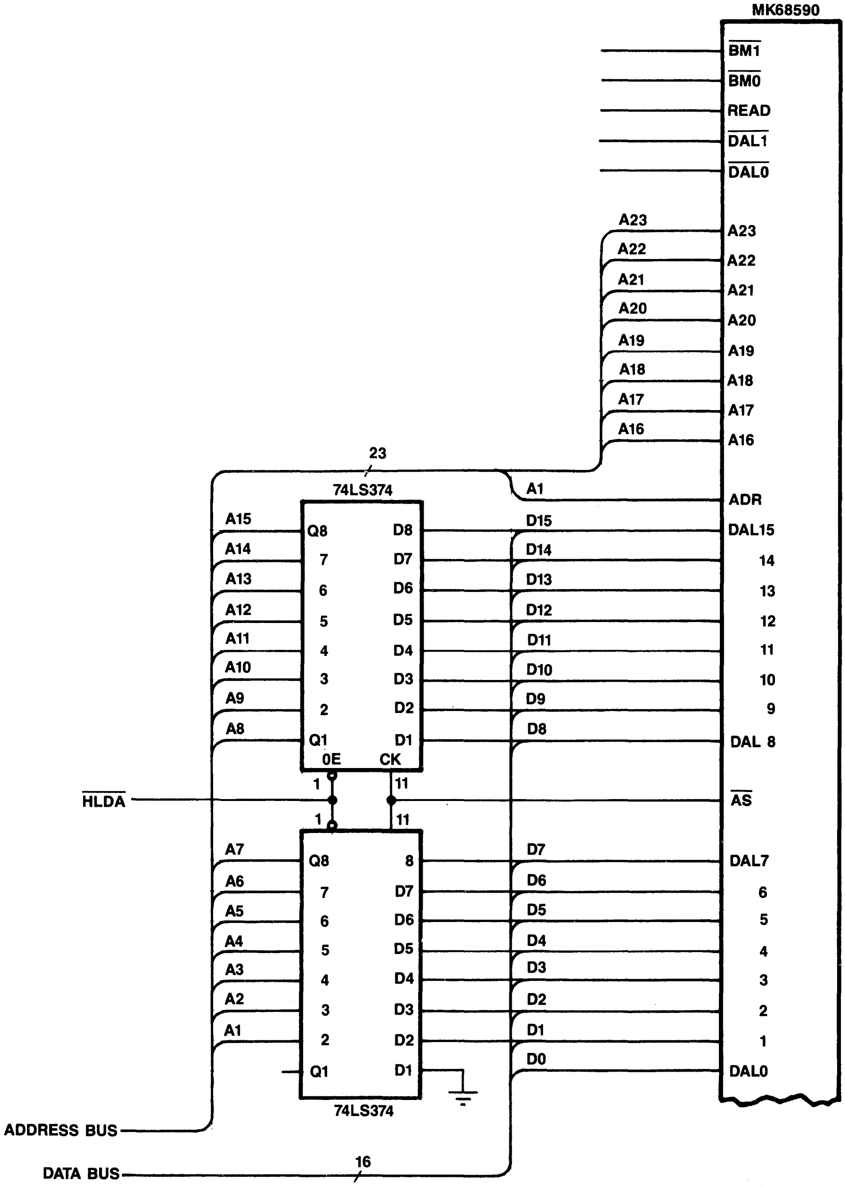


Figure 14. Address/Data Bus Interface Circuit

6.5 INTERRUPT CIRCUITRY

The interrupt circuitry, as shown in Figure 15, handles incoming interrupts to the MK68000, and also handles the interrupt acknowledgement scheme. This circuit has two interrupt sources. One can come from the LANCE, the other can come from a push button on the breadboard. The push button informs the MK68000 that a message is ready to be transmitted.

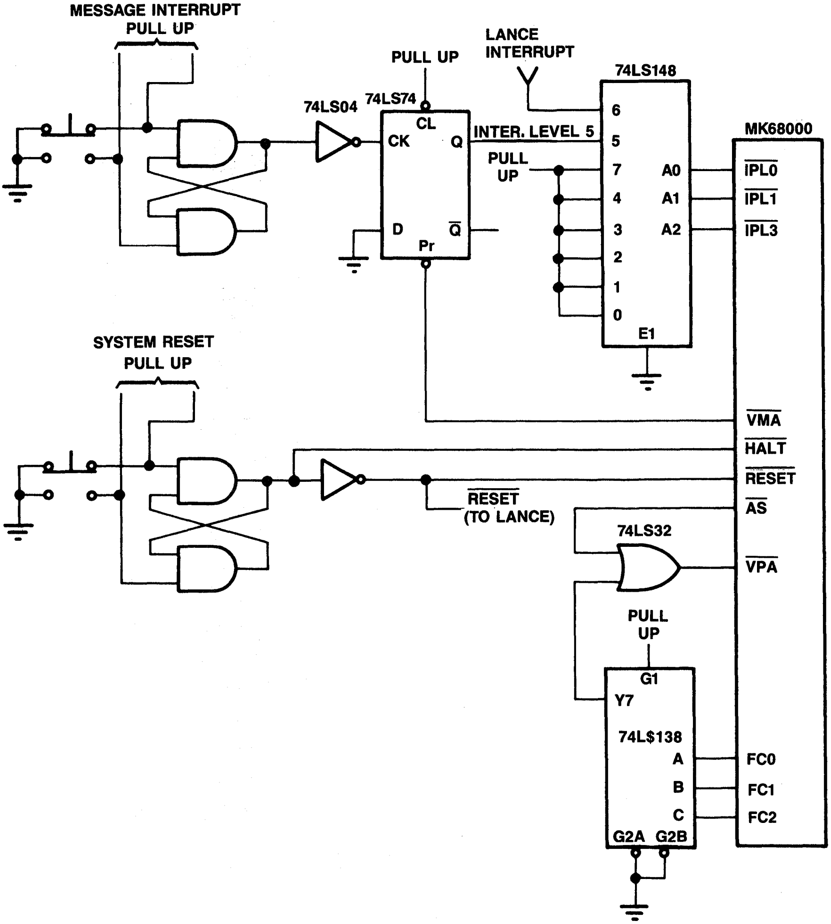


Figure 15. Interrupt and Auto Vector Circuit

6.6 AUTOVECTORING CIRCUIT

Interrupts in this program are handled with autovectoring. The interrupt from the LANCE is set to interrupt level six, and the interrupt that represents a "message ready to be transferred" from the system host is set at interrupt level five. Level seven interrupt is the nonmaskable interrupt, and is not used in this circuit.

Upon an interrupt, the MK68000 responds with binary "111'" at the Function Code Output (FC0-2). This is decoded and Valid Peripheral Acknowledge (VPA) is generated from it, which tells the MK68000 to autovector (see the MK68000 Users Manual for more details on autovectoring).

6.7 BUS ARBITRATION CIRCUIT

The Bus Arbitration circuit, as shown in Figure 16, generates the necessary control and handshake signals needed for the LANCE to take control of the bus. The MK68000 has three handshake signals for bus arbitration, while the LANCE has only two. When the LANCE requires control of the bus, it asserts $\overline{\text{HOLD}}$. The MK68000 returns a Bus Grant ($\overline{\text{BG}}$) signal. $\overline{\text{BG}}$ along with $\overline{\text{DTACK}}$ and $\overline{\text{AS}}$ generate hold acknowledge HLDA and Bus grant acknowledge ($\overline{\text{BGACK}}$). $\overline{\text{DTACK}}$ and $\overline{\text{AS}}$ are needed to verify that the previous cycle is over before the LANCE takes control of the bus. A timing diagram of the bus arbitration process is shown in Figure 17.

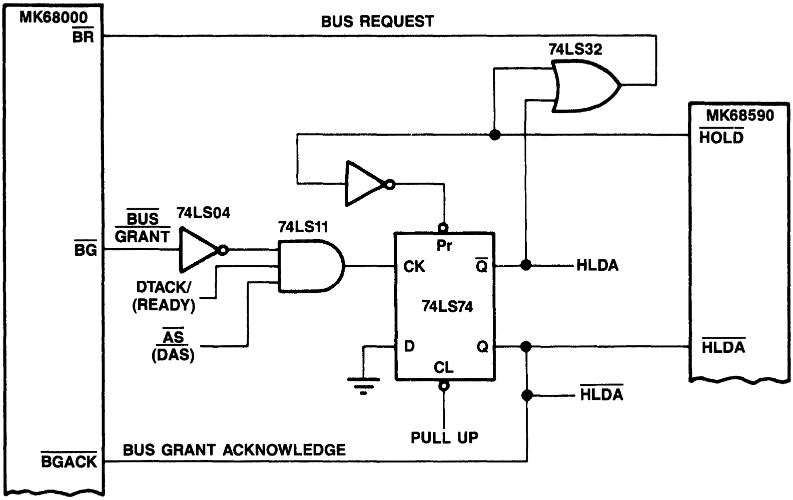


Figure 16. Bus Arbitration Circuit

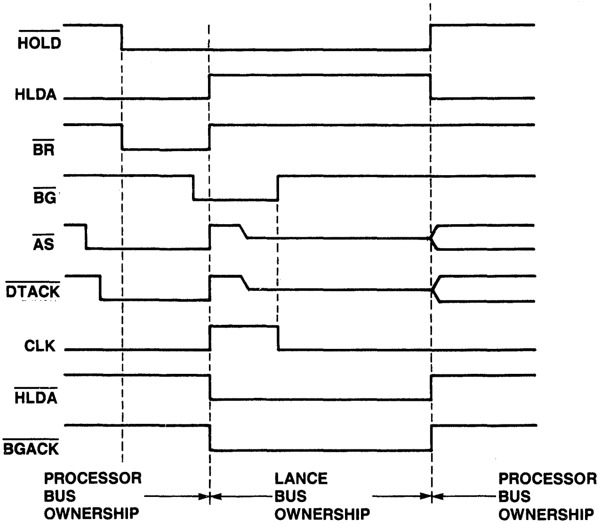


Figure 17. Bus Arbitration Timing

6.8 SIA-LANCE INTERCONNECT

Users must carefully plan the interconnect layout between SIA, the LANCE, and the output connector to the transceiver. The SIA is a very fast ECL interface device. It has an internal voltage controlled oscillator (VCO) that generates a 40 MHz clock from which internal timing is derived. This device is very susceptible to noise. For this reason, the proximity of the SIA to the LANCE and output connector is important. Very close coupling between all filter and terminating devices is required. They must be mounted directly adjacent to the SIA pins in printed circuit boards and must be attached directly to the socket pin in a breadboard situation.

Only low-profile sockets should be used for the SIA in a breadboard situation. ZIP-DIPs, or high-profile sockets result in an environment too noisy for the fast communication rates these devices generate.

Decoupling is also very important when building a breadboard or laying out a PC board. The Ground and V_{CC} pins (Pins 1 and 48) of the LANCE should be decoupled to reduce noise possibilities. These capacitors should be attached directly to the socket pins in a breadboard situation. Figure 18 gives suggested filter values to be used on the SIA.

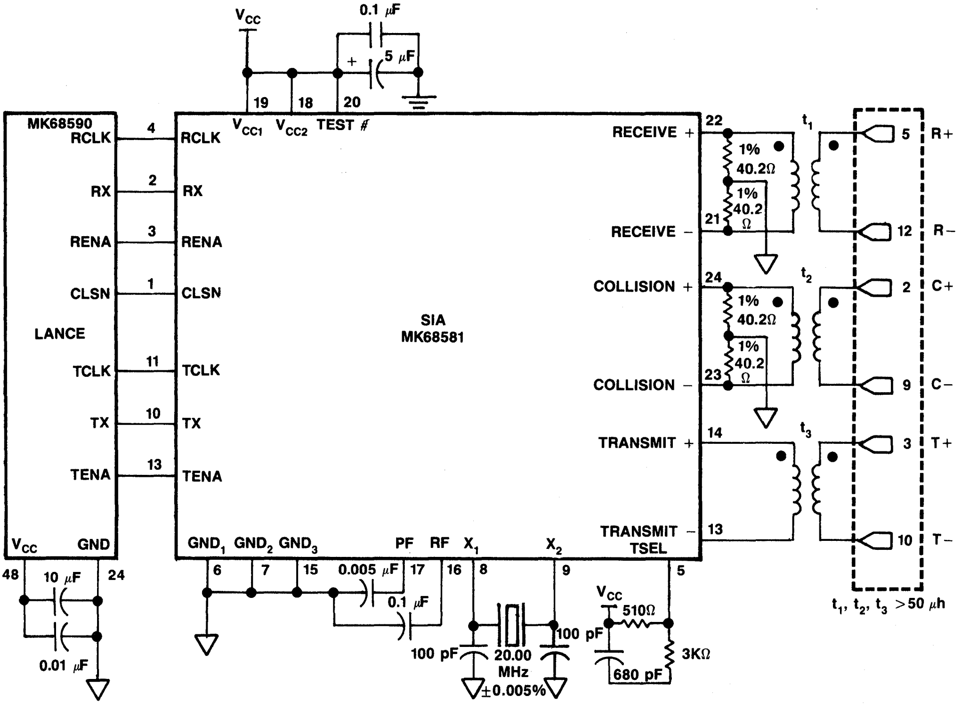


Figure 18. SIA Filter Values

7.0. SOFTWARE DESCRIPTION

7.1 INTRODUCTION

The software needed to generate a basic interface between the LANCE and the MK68000 is described in three different formats: a written description of the step-by-step process, flow charts of the individual submodules, and a printout of the actual assembly code. The assembly code appears in Appendices A, B, and C.

Users should read the software description along with studying the flow charts and assembly code. This triple reinforcement should make it easier to comprehend software requirements.

The software has four basic modules, as shown in the memory map of Figure 19. They include: Initialization & Diagnostics, LANCE Interrupt, Message Interrupt, and Status Module.

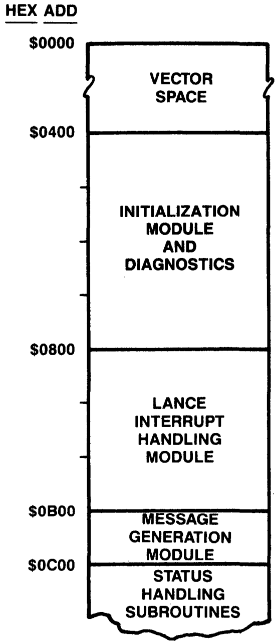

HEX ADD
$0000
VECTOR SPACE
$0400
INITIALIZATION MODULE AND DIAGNOSTICS
$0800
LANCE INTERRUPT HANDLING MODULE
$0B00
MESSAGE GENERATION MODULE
$0C00
STATUS HANDLING SUBROUTINES

Figure 19. Software Memory Map

7.2 INITIALIZATION & DIAGNOSTICS SOFTWARE MODULE

7.2.1 INTRODUCTION

The Initialization & Diagnostics Module, as shown in Figure 20, contains six submodules:

1. Clear
2. Block-Move
3. Receive Ring Initialization
4. Transmit Ring Initialize
5. Diagnostics
6. Normal Initialize.

The diagnostics portion is actually a subroutine. Diagnostics may be performed at any time by simpling calling this subroutine.

In addition to these six submodules are two subroutines: Control and status Register Initialization Subroutine, and the Cyclic Redundancy Check Subroutine.

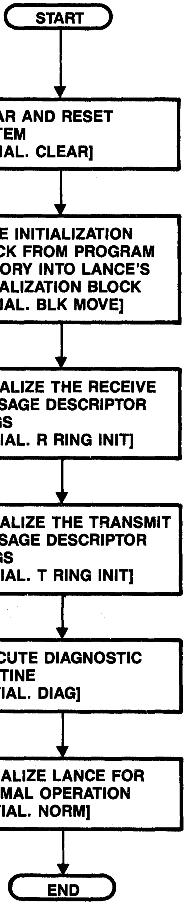

START
CLEAR AND RESET SYSTEM [INITIAL. CLEAR]
MOVE INITIALIZATION BLOCK FROM PROGRAM MEMORY INTO LANCE'S INITIALIZATION BLOCK [INITIAL. BLK MOVE]
INITIALIZE THE RECEIVE MESSAGE DESCRIPTOR RINGS [INITIAL. R RING INIT]
INITIALIZE THE TRANSMIT MESSAGE DESCRIPTOR RINGS [INITIAL. T RING INIT]
EXECUTE DIAGNOSTIC ROUTINE [INITIAL. DIAG]
INITIALIZE LANCE FOR NORMAL OPERATION [INITIAL. NORM]
END

Figure 20. Initialization Software Module [LANCE. INITIAL]

7.2.2 CLEAR SUBMODULE

Upon powerup, the MK68000 (also referred to as the local host, or host) addresses the reset vector location $0000 that holds the address of the start of the program. The starting address of the Clear Submodule, shown in Figure 21, is at location $400.

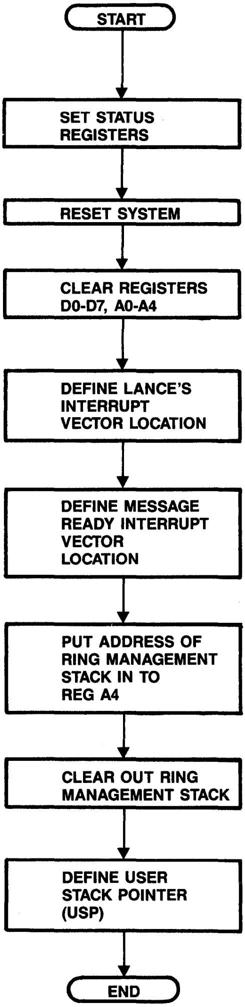

START
SET STATUS REGISTERS
RESET SYSTEM
CLEAR REGISTERS D0-D7, A0-A4
DEFINE LANCE'S INTERRUPT VECTOR LOCATION
DEFINE MESSAGE READY INTERRUPT VECTOR LOCATION
PUT ADDRESS OF RING MANAGEMENT STACK IN TO REG A4
CLEAR OUT RING MANAGEMENT STACK
DEFINE USER STACK POINTER (USP)
END

Figure 21. Clear Submodule [LANCE. INITIAL. CLEAR]

The program's first action is to reset the LANCE and other peripherals on the local system bus. Next, it clears out the address and data registers, sets the interrupt mask, loads the interrupt autovectors and both system and ring-management stack locations. The ring management area starting address is stored in Address Register A4. Following this action, the host clears all data held in the ring status register.

7.2.3 BLOCK-MOVE SUBMODULE

Once the registers have all been cleared, the processor's next task is to move the LANCE's initialization block into memory where the LANCE can access it. Information also must be extracted from the initialization block which allows the software to determine the base addresses of the transmit and receive descriptor rings and their respective lengths. These operations take place in the "Block-Move" submodule described by the flowchart in Figure 22.

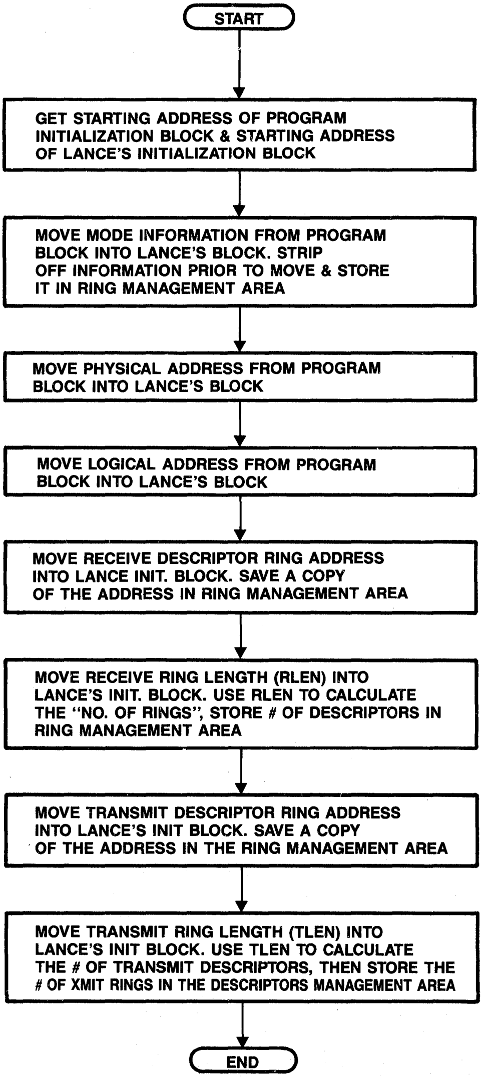

START
GET STARTING ADDRESS OF PROGRAM INITIALIZATION BLOCK & STARTING ADDRESS OF LANCE'S INITIALIZATION BLOCK
MOVE MODE INFORMATION FROM PROGRAM BLOCK INTO LANCE'S BLOCK. STRIP OFF INFORMATION PRIOR TO MOVE & STORE IT IN RING MANAGEMENT AREA
MOVE PHYSICAL ADDRESS FROM PROGRAM BLOCK INTO LANCE'S BLOCK
MOVE LOGICAL ADDRESS FROM PROGRAM BLOCK INTO LANCE'S BLOCK
MOVE RECEIVE DESCRIPTOR RING ADDRESS INTO LANCE INIT. BLOCK. SAVE A COPY OF THE ADDRESS IN RING MANAGEMENT AREA
MOVE RECEIVE RING LENGTH (RLEN) INTO LANCE'S INIT. BLOCK. USE RLEN TO CALCULATE THE "NO. OF RINGS", STORE # OF DESCRIPTORS IN RING MANAGEMENT AREA
MOVE TRANSMIT DESCRIPTOR RING ADDRESS INTO LANCE'S INIT BLOCK. SAVE A COPY OF THE ADDRESS IN THE RING MANAGEMENT AREA
MOVE TRANSMIT RING LENGTH (TLEN) INTO LANCE'S INIT BLOCK. USE TLEN TO CALCULATE THE # OF TRANSMIT DESCRIPTORS, THEN STORE THE # OF XMIT RINGS IN THE DESCRIPTORS MANAGEMENT AREA
END

Figure 22. Block-Move Submodule [LANCE, INITIAL. BLKMOVE]

The LANCE buffer management information is set up as equates. Equates define the following information:

1. Starting address of the initialization block
2. Displacement needed between the initialization block's starting address to the receiver and transmit descriptors' starting address
3. Number of transmit and receive buffers desired
4. Desired buffer size.

By altering one or more equate statements, any of these buffer management areas may be relocated in memory.

This software routine makes all calculations needed to initialize the receive and transmit descriptors. The only limitation is that all receive descriptors must be in a contiguous block of memory. The same holds true for transmit descriptors. The buffers for the receive and transmit operation may be placed anywhere throughout the memory by specifying a buffer displacement from the descriptor's starting address (see Figure 23).

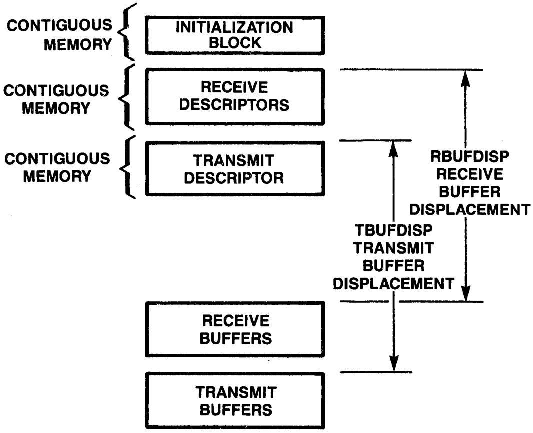


Figure 23. Buffer Displacement Diagram

This design was chosen to give the program more flexibility. The same piece of software can be used, independent of buffer number, size, and location.

Information is stripped by the block-move submodule from the initialization block source and transferred to the LANCE shared memory. If the source is EPROM, the initialization block is identical upon each powerup. If the source is downloaded from the main host, the initialization block may vary from one powerup sequence to the next. The software in this Application Note was written so the initialization block, receive and transmit descriptors, and the receive and transmit buffers are all in contiguous memory.

The "Block-Move" submodule also strips the receive and transmit ring lengths as it moves the initialization block into memory. It stores the values to be used in memory allocation calculations in the receive and transmit ring initialization routines. It also extracts mode information and sets the corresponding bits in the ring management status register. In addition, it also copies, reformats for MK68000 compatability, and stores the receive and transmit ring base address for later use.

In this software design, the receive and transmit descriptor base addresses are predetermined and the program generates the buffer addresses from ring length information. The software also could have been structured to give only ring length information. The program would then determine ring and buffer locations. Another structure may have all address information pre-calculated without any calculations necessary upon initialization, although this structure would not allow flexible ring management. The structure depends on the application desired.

7.2.4 RECEIVE RING INITIALIZATION SUBMODULE

Once the entire initialization block is moved into the share memory, the "Block-Move" submodule is complete and the "Receive Ring Initialization" submodule begins. This submodule initializes all receive ring descriptors by generating Receive Descriptors 0 through 3 (RMDO, RMD1, RMD2, & RMD3), as shown in Figure 24. It generates each descriptor's respective buffer address by displacing it with the buffer length and displacement.

For example, if 256-byte buffers are required and the receive descriptor has a starting address of hex $5000 and a buffer displacement of hex $1000, users would first add the buffer displacement to the starting address of the descriptors. This would give a receive buffer starting address of hex $6000. Therefore, the buffer address of receive buffer number 1 would be $6000. Receive buffer number 2 would have a starting address of $6100, Receive number 3 would have a starting address of $6200 and so on, each displaced 256 bytes, or one buffer length from the previous buffer. Once the descriptor is initialized, ownership of it is given to the LANCE to be used when an incoming message is received.

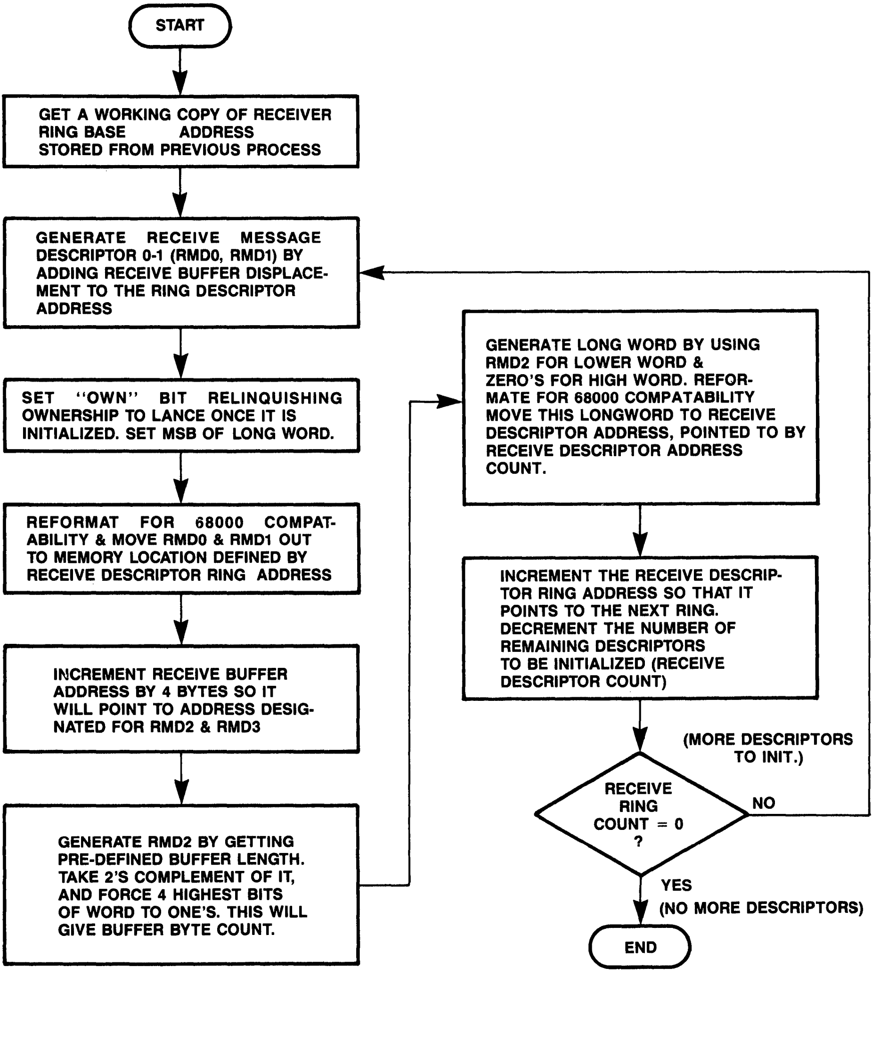

START
GET A WORKING COPY OF RECEIVER RING BASE ADDRESS STORED FROM PREVIOUS PROCESS
GENERATE RECEIVE MESSAGE DESCRIPTOR 0-1 (RMD0, RMD1) BY ADDING RECEIVE BUFFER DISPLACEMENT TO THE RING DESCRIPTOR ADDRESS
SET "OWN" BIT RELINQUISHING OWNERSHIP TO LANCE ONCE IT IS INITIALIZED. SET MSB OF LONG WORD.
REFORMAT FOR 68000 COMPATABILITY & MOVE RMD0 & RMD1 OUT TO MEMORY LOCATION DEFINED BY RECEIVE DESCRIPTOR RING ADDRESS
INCREMENT RECEIVE BUFFER ADDRESS BY 4 BYTES SO IT WILL POINT TO ADDRESS DESIGNATED FOR RMD2 & RMD3
GENERATE RMD2 BY GETTING PRE-DEFINED BUFFER LENGTH. TAKE 2'S COMPLEMENT OF IT, AND FORCE 4 HIGHEST BITS OF WORD TO ONE'S. THIS WILL GIVE BUFFER BYTE COUNT.
GENERATE LONG WORD BY USING RMD2 FOR LOWER WORD & ZERO'S FOR HIGH WORD. REFORMATE FOR 68000 COMPATABILITY MOVE THIS LONGWORD TO RECEIVE DESCRIPTOR ADDRESS, POINTED TO BY RECEIVE DESCRIPTOR ADDRESS COUNT.
INCREMENT THE RECEIVE DESCRIPTOR RING ADDRESS SO THAT IT POINTS TO THE NEXT RING. DECREMENT THE NUMBER OF REMAINING DESCRIPTORS TO BE INITIALIZED (RECEIVE DESCRIPTOR COUNT)
(MORE DESCRIPTORS TO INIT.)
RECEIVE RING COUNT = 0 ?
NO
YES
(NO MORE DESCRIPTORS)
END

Figure 24. Receive Ring Initialization Submodule [LANCE. INITIAL. RRINGINIT]

7.2.5 TRANSMIT RING INITIALIZATION SUBMODULE

When all receive ring descriptors are initialized, the program proceeds into the "Transmit Ring Initialization" submodule shown in Figure 25. This submodule initializes the transmit descriptors in much the same way it initializes the receive descriptors, with the exception that the host retains ownership of the transmit descriptors. They are used during message transmission. This submodule also generates the 2's complement of the byte count (BCNT) and places it in the address specified for TMD2.

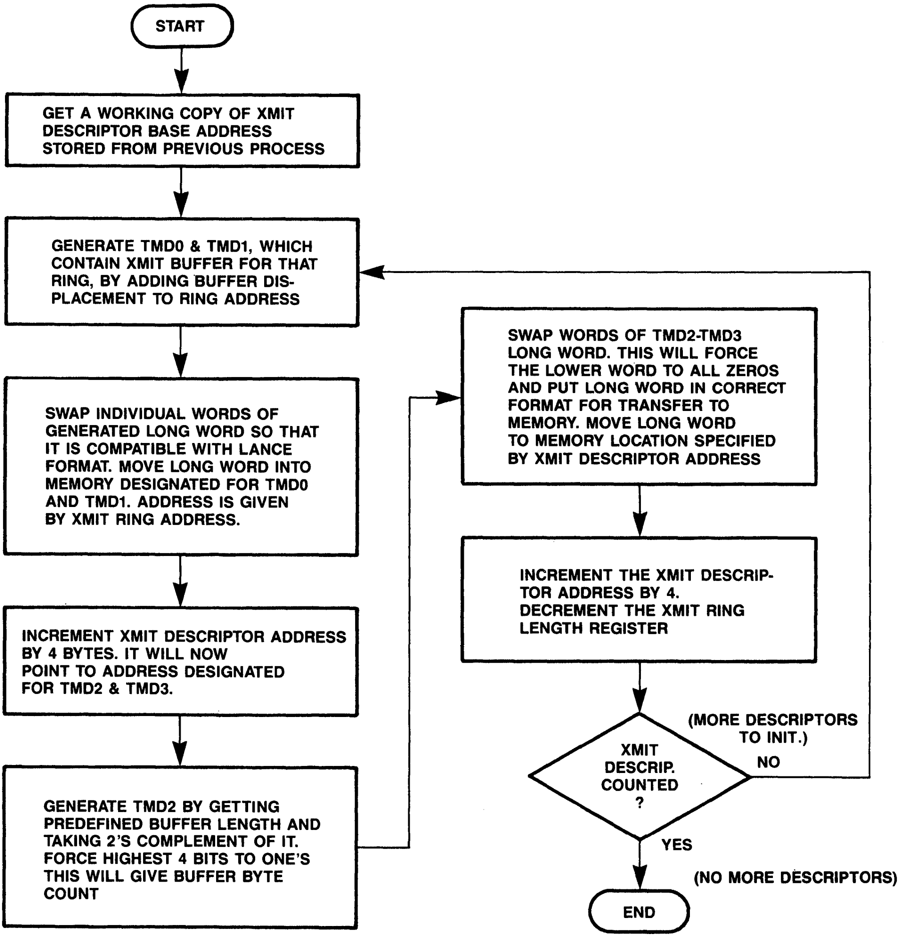

START
GET A WORKING COPY OF XMIT DESCRIPTOR BASE ADDRESS STORED FROM PREVIOUS PROCESS
GENERATE TMD0 & TMD1, WHICH CONTAIN XMIT BUFFER FOR THAT RING, BY ADDING BUFFER DISPLACEMENT TO RING ADDRESS
SWAP INDIVIDUAL WORDS OF GENERATED LONG WORD SO THAT IT IS COMPATIBLE WITH LANCE FORMAT. MOVE LONG WORD INTO MEMORY DESIGNATED FOR TMD0 AND TMD1. ADDRESS IS GIVEN BY XMIT RING ADDRESS.
INCREMENT XMIT DESCRIPTOR ADDRESS BY 4 BYTES. IT WILL NOW POINT TO ADDRESS DESIGNATED FOR TMD2 & TMD3.
GENERATE TMD2 BY GETTING PREDEFINED BUFFER LENGTH AND TAKING 2'S COMPLEMENT OF IT. FORCE HIGHEST 4 BITS TO ONE'S THIS WILL GIVE BUFFER BYTE COUNT
SWAP WORDS OF TMD2-TMD3 LONG WORD. THIS WILL FORCE THE LOWER WORD TO ALL ZEROS AND PUT LONG WORD IN CORRECT FORMAT FOR TRANSFER TO MEMORY. MOVE LONG WORD TO MEMORY LOCATION SPECIFIED BY XMIT DESCRIPTOR ADDRESS
INCREMENT THE XMIT DESCRIPTOR ADDRESS BY 4. DECREMENT THE XMIT RING LENGTH REGISTER
XMIT DESCRIP. COUNTED ?
(MORE DESCRIPTORS TO INIT.)
NO
YES
(NO MORE DESCRIPTORS)
END

Figure 25. Transmit Ring Initialization Submodule [LANCE. INITIAL. TRINGINIT]

7.2.6 DIAGNOSTIC SUBROUTINE

At this point in the program, the initialization block is in memory, which the LANCE can access. The receive and transmit rings are initialized. A diagnostic routine now needs to be run to determine if the LANCE and associated hardware are operating properly. (See Figures 26 and 27.) This is done by placing the LANCE in four different loopback modes:

1. Internal loopback with transmit CRC enabled
2. Internal loopback with transmit CRC disabled
3. Internal loopback with transmit CRC enabled and the collision force bit set.
4. External loopback.

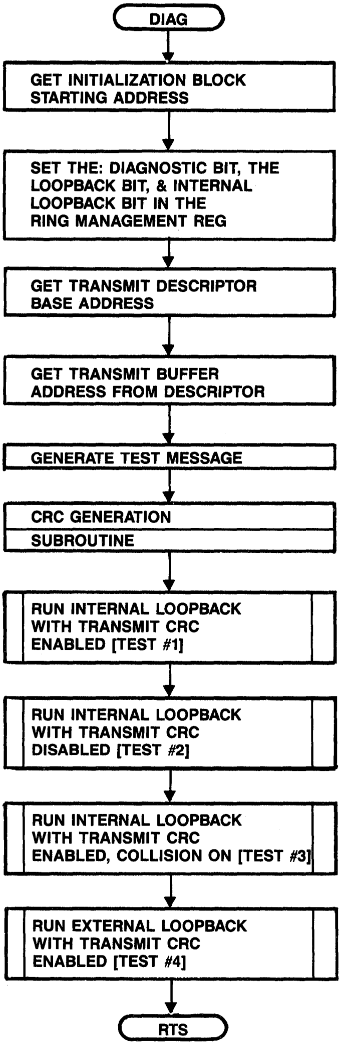

DIAG
GET INITIALIZATION BLOCK STARTING ADDRESS
SET THE: DIAGNOSTIC BIT, THE LOOPBACK BIT, & INTERNAL LOOPBACK BIT IN THE RING MANAGEMENT REG
GET TRANSMIT DESCRIPTOR BASE ADDRESS
GET TRANSMIT BUFFER ADDRESS FROM DESCRIPTOR
GENERATE TEST MESSAGE
CRC GENERATION SUBROUTINE
RUN INTERNAL LOOPBACK WITH TRANSMIT CRC ENABLED [TEST #1]
RUN INTERNAL LOOPBACK WITH TRANSMIT CRC DISABLED [TEST #2]
RUN INTERNAL LOOPBACK WITH TRANSMIT CRC ENABLED, COLLISION ON [TEST #3]
RUN EXTERNAL LOOPBACK WITH TRANSMIT CRC ENABLED [TEST #4]
RTS

Figure 26. Diagnostics' Subroutine [LANCE. INITIAL. DIAG]

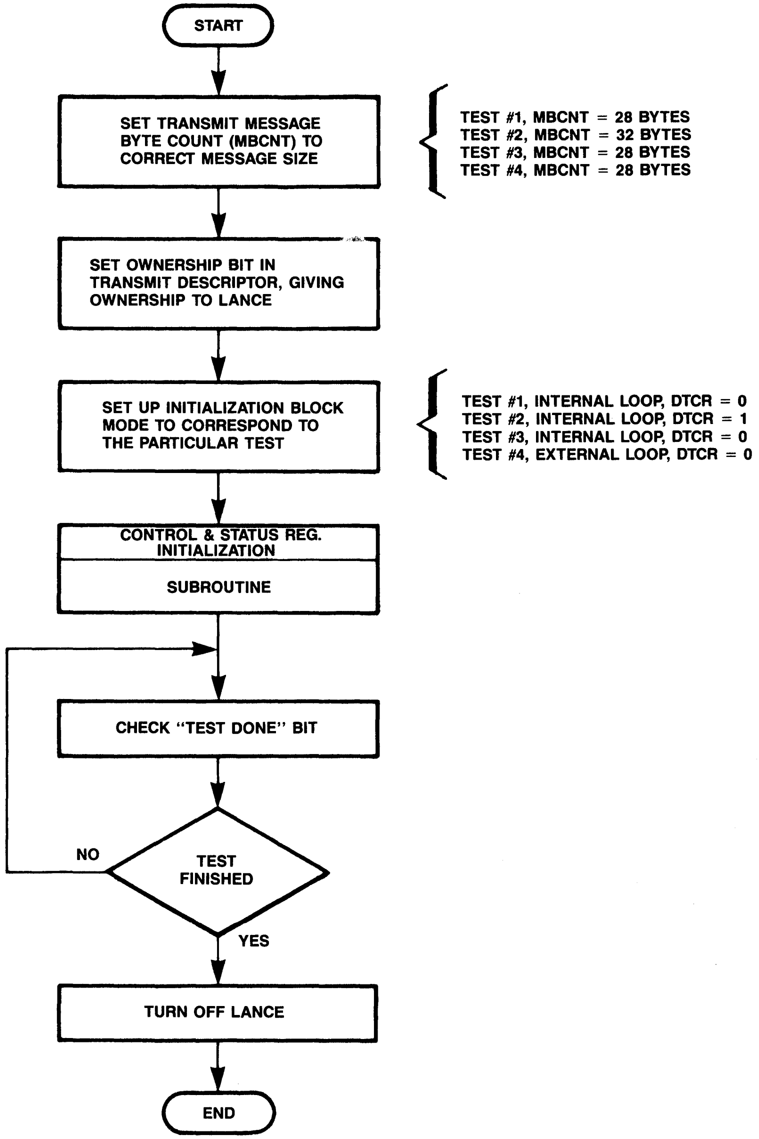

START
SET TRANSMIT MESSAGE BYTE COUNT (MBCNT) TO CORRECT MESSAGE SIZE
TEST #1, MBCNT = 28 BYTES
TEST #2, MBCNT = 32 BYTES
TEST #3, MBCNT = 28 BYTES
TEST #4, MBCNT = 28 BYTES
SET OWNERSHIP BIT IN TRANSMIT DESCRIPTOR, GIVING OWNERSHIP TO LANCE
SET UP INITIALIZATION BLOCK MODE TO CORRESPOND TO THE PARTICULAR TEST
TEST #1, INTERNAL LOOP, DTCR = 0
TEST #2, INTERNAL LOOP, DTCR = 1
TEST #3, INTERNAL LOOP, DTCR = 0
TEST #4, EXTERNAL LOOP, DTCR = 0
CONTROL & STATUS REG. INITIALIZATION
SUBROUTINE
CHECK "TEST DONE" BIT
NO
TEST FINISHED
YES
TURN OFF LANCE
END

Figure 27. Loopback Diagnostic Test Routine

To test the LANCE, a test message must be generated and placed in the first transmit buffer. In addition, the first transmit descriptor must be initialized.

The test data message created in this program is 28 bytes long. The test message is all hex A's, which is alternating binary 1's and 0's. The maximum amount of data that can be transmitted in loopback mode, whether external or internal, is 32 bytes. With transmit CRC enabled, a 28-byte message is transmitted but, the actual message size is 32 bytes because the LANCE tags four bytes of hardware CRC upon transmission. The transmit length constraint is due to the size limitation of the LANCE's SILO.

In this Application Note, the loopback date size is 28 bytes for all four tests.

After the message is generated, a CRC is generated by calling the software CRC subroutine described in Section 7.2.6.1. In all but the second loopback test, the LANCE's transmit CRC is enabled (bit 3 of the mode is set for transmit CRC disable). When the transmit CRC is disabled, a software CRC is generated and transmitted with the message. For the remaining loopback tests, the LANCE generates the CRC code in hardware and tags it on the end of the transmitted message. This hardware-generated CRC is then compared to the software CRC to assure proper operation of the LANCE.

The CRC generation is followed by initialization of the transmit descriptor. The buffer byte count (BCNT) of 28 is written into transmit message descriptor 2 (TMD2). The start and end of packet bits, as well as the own bit are set in transmit message descriptor 1 (TMD1). Following this descriptor initialization, the mode is set to: promiscuous, internal loopback, with the "disable transmit CRC" bit set to "0".

As described before, with transmit CRC enabled, a CRC is generated for the outgoing message, but is not checked for the incoming message. This is because the LANCE has only one CRC device. The device can generate the CRC or check the CRC, but not concurrently. Verification of proper CRC occurs in the Receive Interrupt submodule, described later.

The final step in each loopback test is an initialization of the LANCE's control and status registers (CSRINIT subroutine call). The CSRINIT subroutine initializes the control and status registers which effectively starts the LANCE. These subroutine steps are summarized in Section 7.2.6.2.

Once the CSRINIT subroutine has executed, the LANCE initializes itself. The program stays in a loop, waiting for completion of this part of the loopback test. This is indicated when a test-complete bit is set in the ring management status register. Once one portion of the loopback test has completed, the next portion begins.

The steps taken after the CSR initialization subroutine is called include:

1. The LANCE is initialized by calling [CSRINIT].
2. The LANCE requests the bus and makes DMA cycles. It reads the entire initialization block.
3. The LANCE polls the receive and transmit rings to check for ownership (see MK68590 LANCE Technical Manual for a more detailed description of polling routines.)
4. Finding that it owns the transmit ring, it enters its transmit DMA routine.
5. Once it has completed its transmission, it immediately starts to DMA the message it concurrently received into the receive buffer.
6. The LANCE then interrupts the MK68000 to notify it that a transmission and reception has occurred.
7. The program vectors off to service the interrupt routine. During the exception processing routine, the application-dependent status subroutines are called if an error flag is found.
8. Once all exception processing is complete, the test done bit is set and a return from the exception processing occurs.
9. The "test-done" bit is constantly checked. Once set, the program continues normal execution, turning off the LANCE and proceeding to the next portion of the diagnostics.

When loopback test number one is complete, the second loopback test proceeds in a similar manner as the first. The only difference between loopback tests is the mode. For clarity, this test is repeated four times in the software rather than tightening the code by inserting four loops.

If the diagnostics detects problems with the LANCE, the status subroutines report the errors by setting a status bit in memory. This Application Note does not deal with error handling, since it is application-dependent and out of the real of this document.

After all four loopback tests have completed, the LANCE is initialized for normal operation.

7.2.6.1 CRC CODE SOFTWARE GENERATION

Before the descriptor initialization, a CRC subroutine, is called to generate a 32-bit cycle redundancy check code (CRC) to be added at the end of the message (see Figure 28).

Since all hex A's is the predetermined data in the message, the CRC can also be predetermined. If the test message differed for each test, an actual rigorus software CRC subroutine would have to be designed. But, in this case, the CRC subroutine simply writes $B1109280 out to the message buffer. This data is the CRC code for the 28-byte test message.

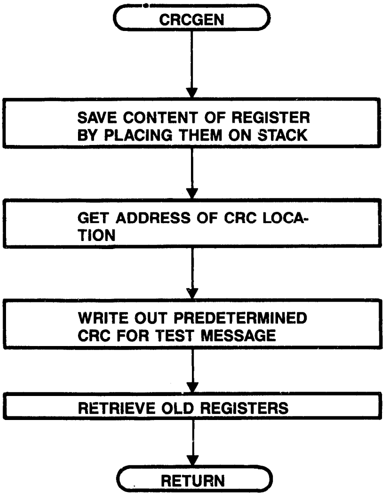

CRCGEN
SAVE CONTENT OF REGISTER BY PLACING THEM ON STACK
GET ADDRESS OF CRC LOCATION
WRITE OUT PREDETERMINED CRC FOR TEST MESSAGE
RETRIEVE OLD REGISTERS
RETURN

Figure 28. Cycle Redundancy Check Generation Subroutine [CRCGEN]

7.2.6.2 CONTROL AND STATUS REGISTER INITIALIZATION SUBROUTINE

The Control and Status register initialization subroutine is shown in Figure 29. The first step is to move the initialization starting address into the LANCE's Control and Status registers. The low order bits <00:15> are placed in CSR1, and the high order bits <16:23> are placed in CSR2. Next, the LANCE is made compatible to the hardware interface by setting BSWP = 1, ACON = 1, and BCON = 0 in CSR3. Finally, a $0043 is written into CSR0, which sets the Interrupt Enable, the Start and Initialize bits. Immediately following this last write, the LANCE starts its initialization procedure by requesting the bus and completing 12 DMA cycles. Each DMA cycle corresponds by moving one word of the initialization block into its internal registers.

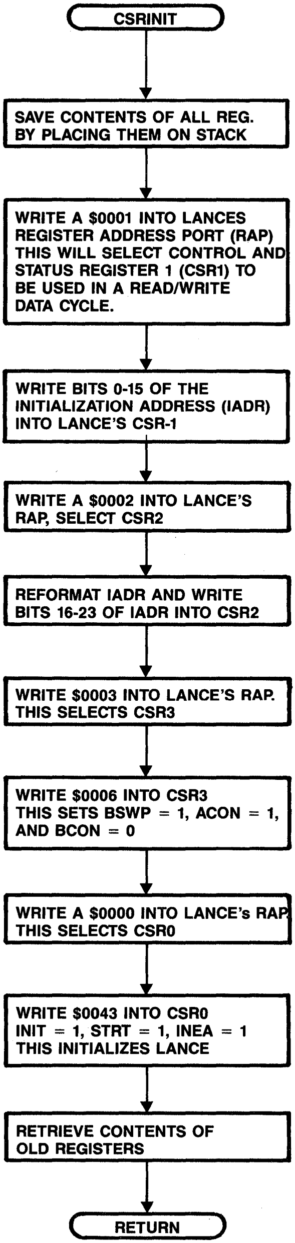

CSRINIT
SAVE CONTENTS OF ALL REG. BY PLACING THEM ON STACK
WRITE A $0001 INTO LANCES REGISTER ADDRESS PORT (RAP) THIS WILL SELECT CONTROL AND STATUS REGISTER 1 (CSR1) TO BE USED IN A READ/WRITE DATA CYCLE.
WRITE BITS 0-15 OF THE INITIALIZATION ADDRESS (IADR) INTO LANCE'S CSR-1
WRITE A $0002 INTO LANCE'S RAP, SELECT CSR2
REFORMAT IADR AND WRITE BITS 16-23 OF IADR INTO CSR2
WRITE $0003 INTO LANCE'S RAP. THIS SELECTS CSR3
WRITE $0006 INTO CSR3 THIS SETS BSWP = 1, ACON = 1, AND BCON = 0
WRITE A $0000 INTO LANCE's RAP THIS SELECTS CSR0
WRITE $0043 INTO CSR0 INIT = 1, STRT = 1, INEA = 1 THIS INITIALIZES LANCE
RETRIEVE CONTENTS OF OLD REGISTERS
RETURN

Figure 29. Control and Status Register Initialization Subroutine [CSRINIT]

Once the DMA cycles are complete, the LANCE interrupts to notify the host of completed initialization. If the Interrupt Enable bit in CSR0 is not set, the LANCE does not interrupt the host, but rather, waits for the host to poll its Control and Status register.

Next, a polling routine is begun to determine which receive and transmit descriptors are owned by LANCE. When a LANCE-owned receive ring is found, that result is stored and polling continues to locate a relinquished transmit ring. The CSR initialization submodule is complete when the final CSR0 takes place. The program then returns from this subroutine.

7.2.7 NORMAL-OPERATION INITIALIZATION

After the diagnostics are complete, the normal-operation initialization occurs. This submodule is shown in Figure 30. The predefined mode in the program initialization block is moved into the LANCE's initialization block. The mode information is checked to determine if the desired mode is a loopback mode. If it is, the ring management status register is set up accordingly. The LANCE is then restarted by calling the control and status register initialization subroutine, CSRINIT.

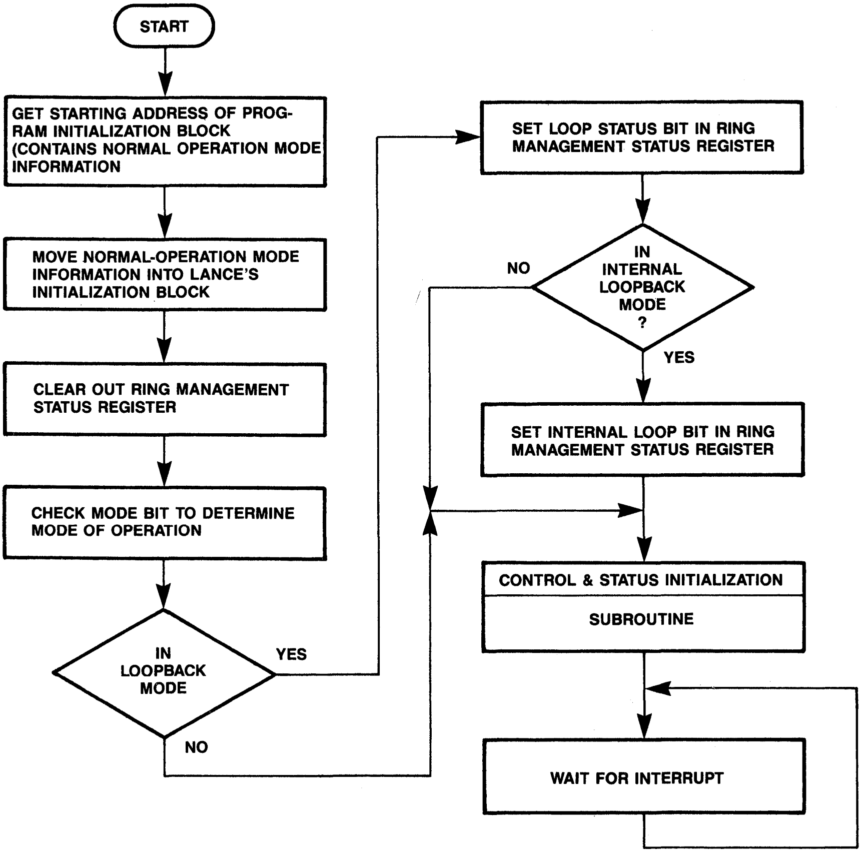

START
GET STARTING ADDRESS OF PROG-RAM INITIALIZATION BLOCK (CONTAINS NORMAL OPERATION MODE INFORMATION
MOVE NORMAL-OPERATION MODE INFORMATION INTO LANCE'S INITIALIZATION BLOCK
CLEAR OUT RING MANAGEMENT STATUS REGISTER
CHECK MODE BIT TO DETERMINE MODE OF OPERATION
IN LOOPBACK MODE
YES
NO
SET LOOP STATUS BIT IN RING MANAGEMENT STATUS REGISTER
IN INTERNAL LOOPBACK MODE ?
NO
YES
SET INTERNAL LOOP BIT IN RING MANAGEMENT STATUS REGISTER
CONTROL & STATUS INITIALIZATION
SUBROUTINE
WAIT FOR INTERRUPT

Figure 30. Normal Operation Initialization Submodule [LANCE. INITIAL. NORM]

After the Normal-Operation Initialization, the initialization software module is complete and stays in a wait loop. The program is completely interrupt-driven from this point on and only moves out of the wait loop when an interrupt occurs.

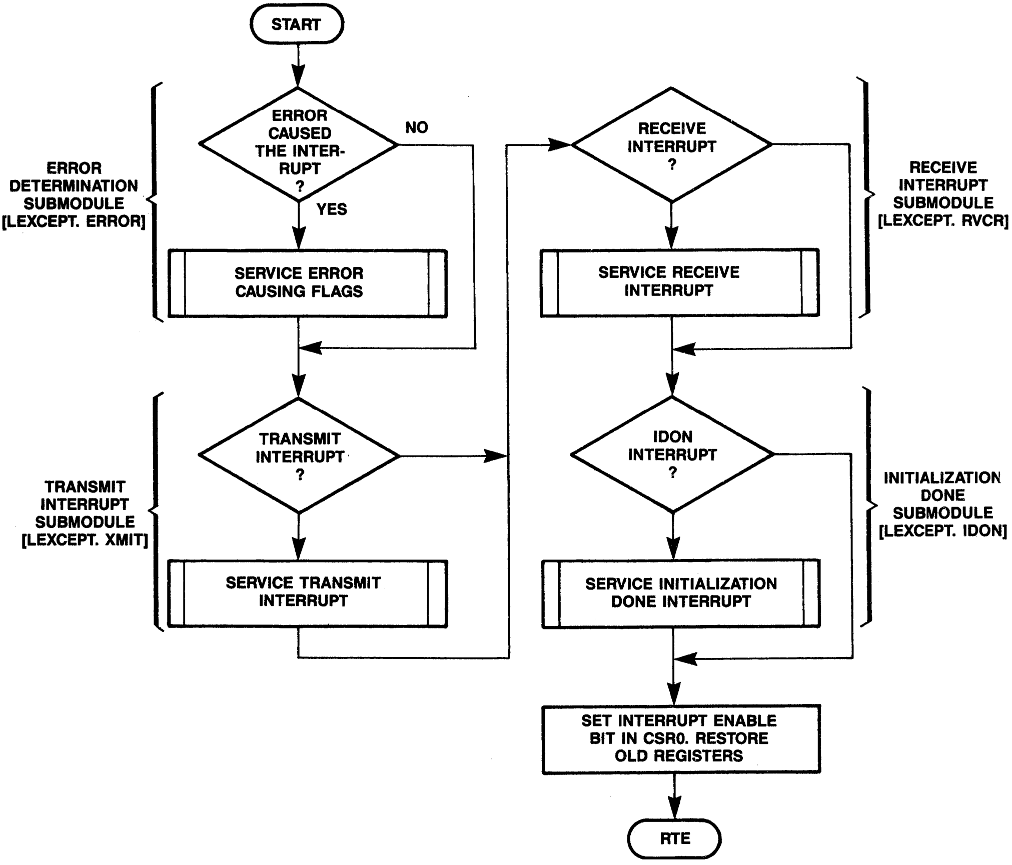
7.3 LANCE INTERRUPT EXCEPTION SOFTWARE MODULE

7.3.1 INTRODUCTION

The LANCE Interrupt Exception Module has basically four submodules. They are:

1. Interrupt Error Determination Submodule
2. Transmit Interrupt Handling Submodule
3. Receive Interrupt Handling Submodule
4. Initialization Done Interrupt Handling Submodule

Figure 31 depicts the flow of the Exception Software Module. The LANCE Interrupt Exception Software Module, also referred to as the LANCE Interrupt Routine, is called when the LANCE interrupts the host processor. The LANCE interrupts the host if a message has been transmitted or received, the LANCE has finished its initialization routine, or an error has occurred.



START
ERROR CAUSED THE INTER-RUPT ?
NO
YES
ERROR DETERMINATION SUBMODULE [LEXCEPT. ERROR]
SERVICE ERROR CAUSING FLAGS
TRANSMIT INTERRUPT ?
TRANSMIT INTERRUPT SUBMODULE [LEXCEPT. XMIT]
SERVICE TRANSMIT INTERRUPT
RECEIVE INTERRUPT ?
RECEIVE INTERRUPT SUBMODULE [LEXCEPT. RVCR]
SERVICE RECEIVE INTERRUPT
IDON INTERRUPT ?
INITIALIZATION DONE SUBMODULE [LEXCEPT. IDON]
SERVICE INITIALIZATION DONE INTERRUPT
SET INTERRUPT ENABLE BIT IN CSR0. RESTORE OLD REGISTERS
RTE

Figure 31. LANCE Exception Processing Module [LEXCEPT]

The LANCE interrupt is hardware set at level 6; the second highest priority. The only higher priority is level seven, that of the nonmaskable interrupt. When the LANCE interrupts the MK68000, it autovectors to memory location $800.

The first action the MK68000 takes is to get the LANCE's status from CSR0. This status helps determine what caused the interrupt. The error bit is the first bit checked. If this bit is not set, the Transmit Interrupt bit is checked to see if a transmitted message caused the interrupt. If an error did cause the interrupt, the software determines the type of error and the routine proceeds to the transmit interrupt check. Once the transmit bit is checked, the receive bit is checked in the same manner. Finally the Initialization Done bit is checked and the program returns to the waiting loop.

All flag-causing conditions are serviced and the error-causing conditions call application-dependent service subroutines. For the purposes of this Application Note, these error service routines simply set certain bits in memory that correspond to the error. The programmer may then examine memory for errors.

7.3.2 INTERRUPT ERROR DETERMINATION SUBMODULE

The first task in this submodule, as shown in Figure 32, is to save the registers. Information in the data and address registers is placed on the stack to be retrieved after exception processing. Next, status information in the Control and Status Register zero (CSR0) is moved into one of the MK68000's registers. Following this, all flags set in CSR0 and the Interrupt Enable bit are cleared. This occurs by clearing bit 6 (Interrupt Enable bit) in the register containing a copy of CSR0, and writing this copy back into CSR0. Since the flags are all cleared by writing a "1" in their bit location, all flags set in CSR0 are cleared. The Error bit in CSR0 clears itself after all the individual error flags are cleared. The interrupt enable bit is cleared by writing a "0" in its bit location. The LANCE's interrupt capabilities are disabled because it is not desirable to have another interrupt occur while the present interrupt from the LANCE is being serviced.

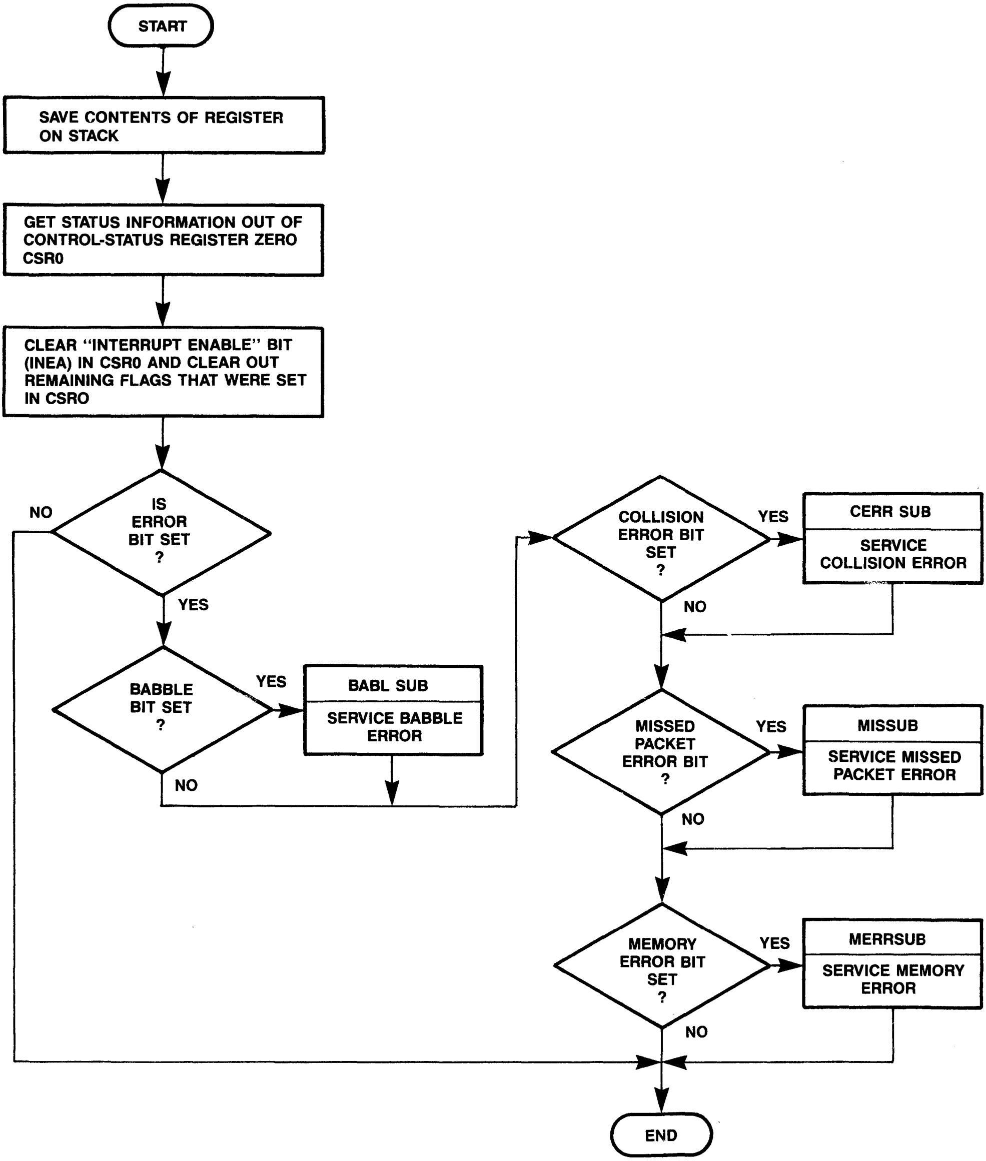

START
SAVE CONTENTS OF REGISTER ON STACK
GET STATUS INFORMATION OUT OF CONTROL-STATUS REGISTER ZERO CSR0
CLEAR "INTERRUPT ENABLE" BIT (INEA) IN CSR0 AND CLEAR OUT REMAINING FLAGS THAT WERE SET IN CSRO
IS ERROR BIT SET ?
NO
YES
BABBLE BIT SET ?
YES
BABL SUB
SERVICE BABBLE ERROR
NO
COLLISION ERROR BIT SET ?
YES
CERR SUB
SERVICE COLLISION ERROR
NO
MISSED PACKET ERROR BIT ?
YES
MISSUB
SERVICE MISSED PACKET ERROR
NO
MEMORY ERROR BIT SET ?
YES
MERRSUB
SERVICE MEMORY ERROR
NO
END

Figure 32. Interrupt Error Determination Submodule [LANCE. LEXCEPT. ERROR]

The Error Summary bit is the first status bit to be checked. If this bit is not set, no error-causing conditions are present and the individual error bits need not be checked. The Transmit Interrupt bit is checked next to see if it is set.

If the Error Summary bit is set, each individual error bit is checked. If any are set, an error-handling subroutine is called to report or service the error. Again, as stated before, these are application-dependent subroutines. The user can implement error handling routines to suit system requirements.

When it is determined whether an error caused the interrupt, the Transmit Interrupt bit is checked. Since more than one condition can cause the interrupt, all bits must be checked. The interrupt pin is simply an OR of the interrupt-causing conditions. If another interrupt occurs after the interrupt pin is asserted, another transition on the interrupt pin will not occur.

7.3.3 TRANSMIT INTERRUPT HANDLING SUBMODULE

If the Transmit Interrupt bit is set, the Transmit Interrupt Handling Submodule is executed (see Figure 33). First, the address pointer to the last transmit descriptor ring is retrieved from the ring management area. This address is incremented by two bytes so it points to the transmit status (TMD1). Next, the transmit status is moved into a MK68000 register. The "More" bit is checked to see if more than one transmission attempts were required. If so, the More counter in the ring management area is updated. The "One" bit is then checked to see if it took exactly one attempt to transmit the message. If so, the One counter is updated in a similar manner.

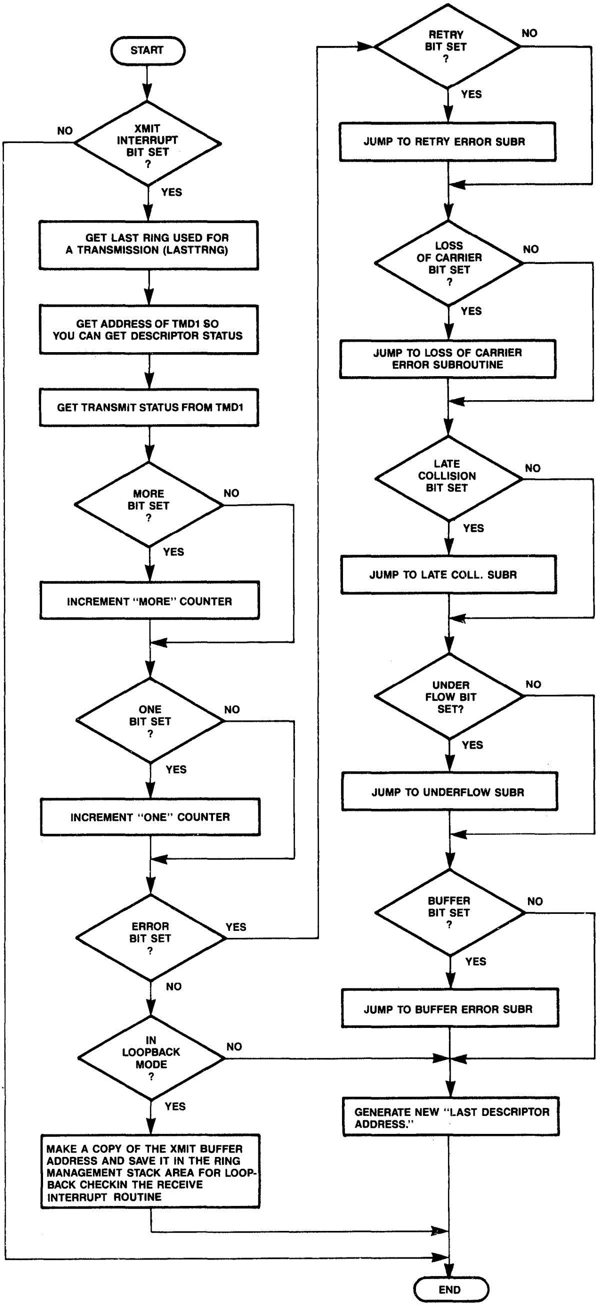

START
XMIT INTERRUPT BIT SET ?
NO
YES
GET LAST RING USED FOR A TRANSMISSION (LASTTRNG)
GET ADDRESS OF TMD1 SO YOU CAN GET DESCRIPTOR STATUS
GET TRANSMIT STATUS FROM TMD1
MORE BIT SET ?
NO
YES
INCREMENT "MORE" COUNTER
ONE BIT SET ?
NO
YES
INCREMENT "ONE" COUNTER
ERROR BIT SET ?
YES
NO
IN LOOPBACK MODE ?
NO
YES
MAKE A COPY OF THE XMIT BUFFER ADDRESS AND SAVE IT IN THE RING MANAGEMENT STACK AREA FOR LOOP-BACK CHECKIN THE RECEIVE INTERRUPT ROUTINE
RETRY BIT SET ?
NO
YES
JUMP TO RETRY ERROR SUBR
LOSS OF CARRIER BIT SET ?
NO
YES
JUMP TO LOSS OF CARRIER ERROR SUBROUTINE
LATE COLLISION BIT SET
NO
YES
JUMP TO LATE COLL. SUBR
UNDER FLOW BIT SET?
NO
YES
JUMP TO UNDERFLOW SUBR
BUFFER BIT SET ?
NO
YES
JUMP TO BUFFER ERROR SUBR
GENERATE NEW "LAST DESCRIPTOR ADDRESS."
END

Figure 33. Transmit Interrupt Handling Submodule [LANCE. LEXCEPT. XMIT]

Next, the Transmit Error bit is tested to see if a transmission error occurred. As with the Interrupt Error bit in CSR0, if this bit is not set, it is not necessary to check each individual error bit, but if this Error Summary bit is set, the program does check each individual error bit. It checks the errors in the following order: retry error, loss of carrier error, late collision error, underflow error, and finally the buffer error. If any of these errors have occurred, the error handling subroutines are called to report and/or service the error-causing conditions.

If the transmission error summary bit is not set, the program checks to see if the LANCE is in the loopback mode. If this is true, a copy of the starting address of the transmit buffer is saved so the data in this buffer can be compared against data in the loopbacked message just received. This comparison indicates loopback mode status.

If the LANCE is in the loopback mode, the pointer to the last transmit descriptor is not incremented. This is because the descriptor buffer must not be available for additional messages until data in the transmit buffer is compared to that in the receive message buffer. This comparison occurs in the Receive Interrupt Handling Submodule of the module that is presently executing. At this time, the Last Transmit Descriptor pointer is updated.

In all other cases, before this submodule is complete, the last descriptor address pointer is updated to point to the next available descriptor. When the last descriptor address is updated, the present address pointed to must be compared with the bottom of the ring.

If the pointer is pointing to the bottom of the ring, it must not be simply incremented but, the address of the top of the ring must be placed in the pointer register. Once the Transmit Interrupt Handling Submodule is complete, the Receive Interrupt bit is checked in the CSR0 status.

7.3.4 RECEIVE INTERRUPT HANDLING SUBMODULE

The Receive Interrupt Handling Submodule, shown in Figure 34, is the most complicated piece of software in this program because it must handle loopback messages differently from a normal received message.

Two loopback situations can occur: One can occur during the diagnostic routine, and the other is the normal condition loopback. The latter loopback happens when the programmer desires to have external or internal loopback as a normal operating mode. This would most likely be the case during debut and system integration.

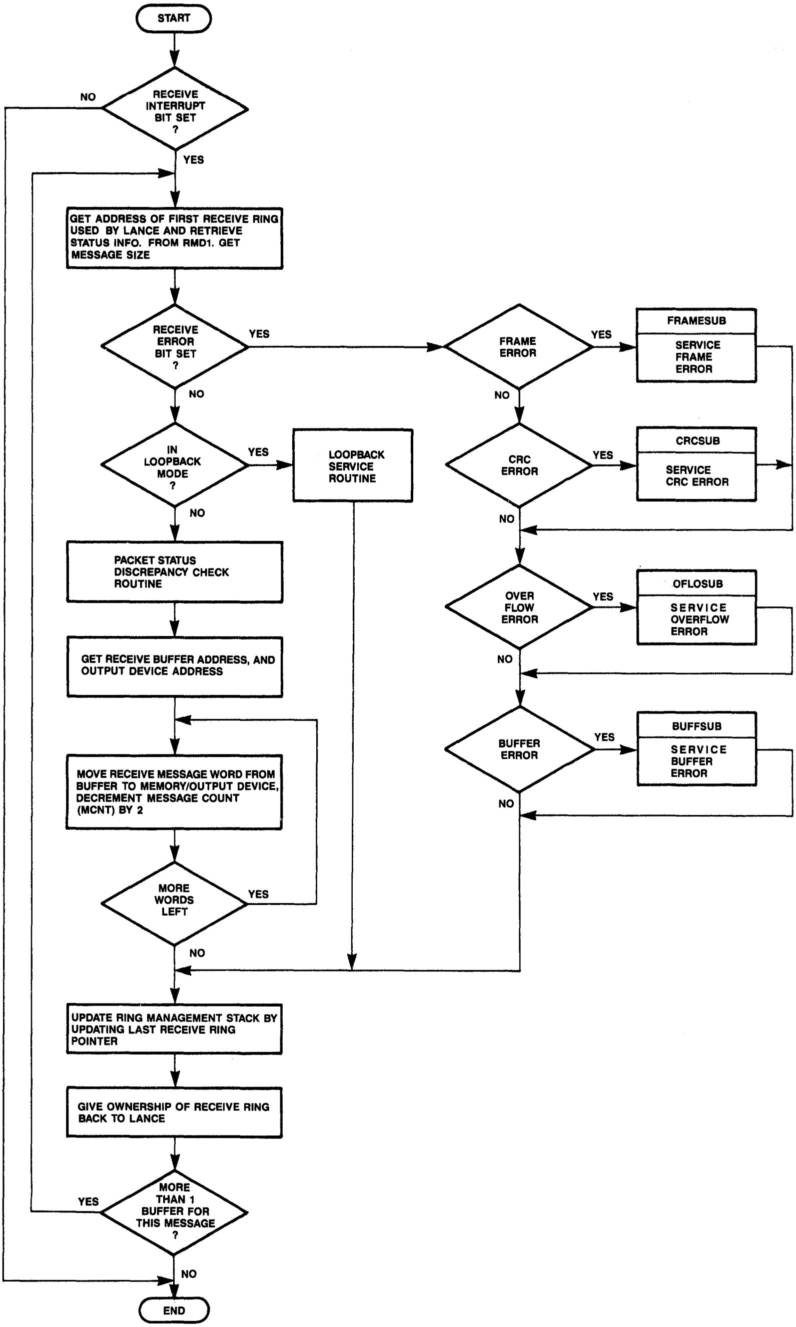

START
RECEIVE INTERRUPT BIT SET ?
NO
YES
GET ADDRESS OF FIRST RECEIVE RING USED BY LANCE AND RETRIEVE STATUS INFO. FROM RMD1. GET MESSAGE SIZE
RECEIVE ERROR BIT SET ?
YES
NO
FRAME ERROR
YES
NO
FRAMESUB
SERVICE FRAME ERROR
IN LOOPBACK MODE ?
YES
NO
LOOPBACK SERVICE ROUTINE
CRC ERROR
YES
NO
CRCSUB
SERVICE CRC ERROR
PACKET STATUS DISCREPANCY CHECK ROUTINE
OVER FLOW ERROR
YES
NO
OFLOSUB
SERVICE OVERFLOW ERROR
GET RECEIVE BUFFER ADDRESS, AND OUTPUT DEVICE ADDRESS
BUFFER ERROR
YES
NO
BUFFSUB
SERVICE BUFFER ERROR
MOVE RECEIVE MESSAGE WORD FROM BUFFER TO MEMORY/OUTPUT DEVICE, DECREMENT MESSAGE COUNT (MCNT) BY 2
MORE WORDS LEFT
YES
NO
UPDATE RING MANAGEMENT STACK BY UPDATING LAST RECEIVE RING POINTER
GIVE OWNERSHIP OF RECEIVE RING BACK TO LANCE
MORE THAN 1 BUFFER FOR THIS MESSAGE ?
YES
NO
END

Figure 34. Receive Interrupt Handling Submodule [LANCE. LEXCEPT. RVCR]

If during CSR0 status checking the Receive Interrupt bit is set, the Receive Interrupt Handling Submodule is executed. The first step is to check the status of the receive message descriptor that the LANCE has turned over to the host by retrieving the Receive Descriptor pointer from the ring management area. Next, receive message descriptor number one (RMD1) is moved into one of the 68000's registers and the Receive Error Summary bit is checked. If this bit is set, the individual error bits are checked to see which error, or combination of errors, caused the error summary bit to be set. Error bits are checked in the following order: frame error, CRC error, overflow error, and buffer error.

If the Error Summary bit is not set, and LANCE is in the loopback mode, the loopback routine is executed (see Figure 35). The loopback routine is still part of the receive interrupt routine, but is listed separately for clarity.

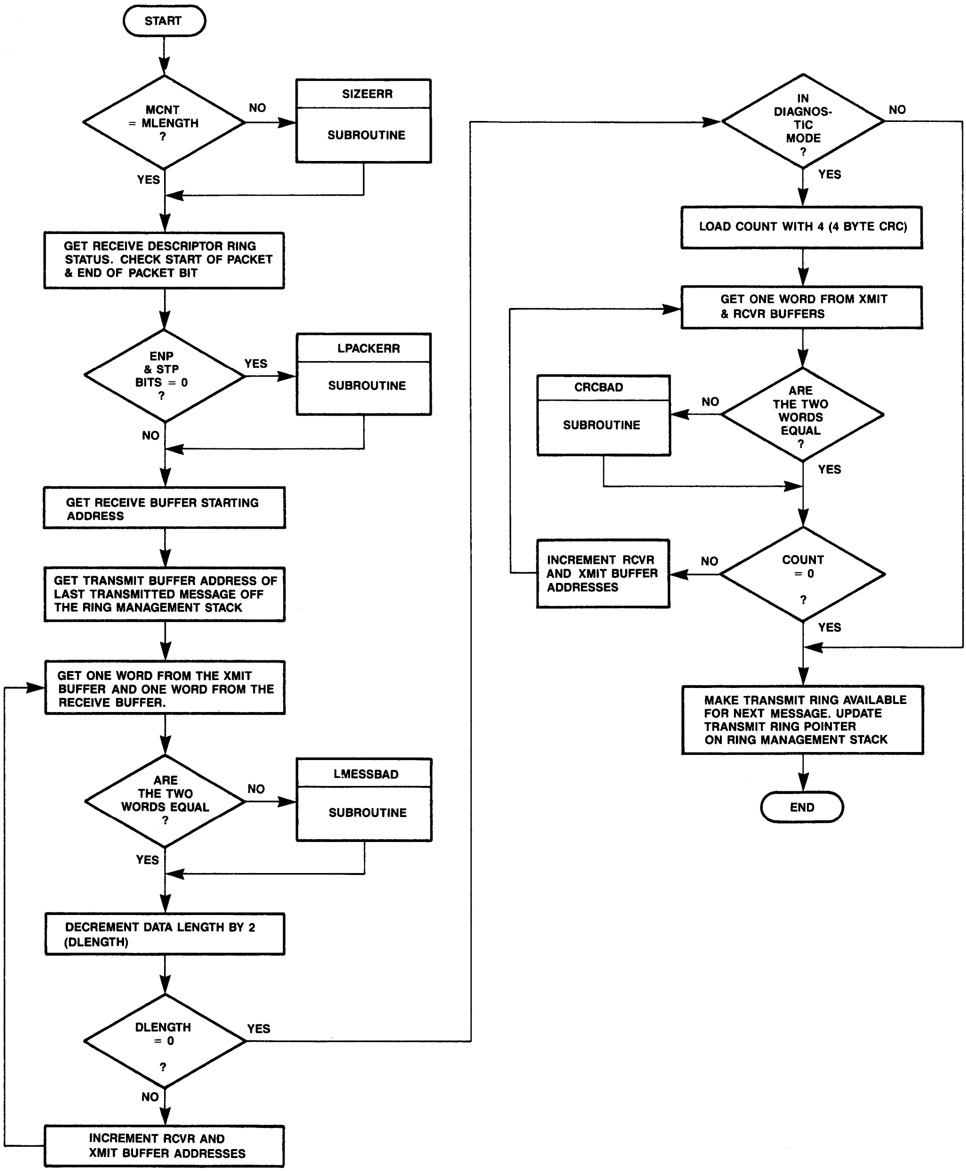

START
MCNT = MLENGTH ?
NO
SIZEERR
SUBROUTINE
YES
GET RECEIVE DESCRIPTOR RING STATUS. CHECK START OF PACKET & END OF PACKET BIT
ENP & STP BITS = 0 ?
YES
LPACKERR
SUBROUTINE
NO
GET RECEIVE BUFFER STARTING ADDRESS
GET TRANSMIT BUFFER ADDRESS OF LAST TRANSMITTED MESSAGE OFF THE RING MANAGEMENT STACK
GET ONE WORD FROM THE XMIT BUFFER AND ONE WORD FROM THE RECEIVE BUFFER.
ARE THE TWO WORDS EQUAL ?
NO
LMESSBAD
SUBROUTINE
YES
DECREMENT DATA LENGTH BY 2 (DLENGTH)
DLENGTH = 0 ?
YES
NO
INCREMENT RCVR AND XMIT BUFFER ADDRESSES
IN DIAGNOS-TIC MODE ?
NO
YES
LOAD COUNT WITH 4 (4 BYTE CRC)
GET ONE WORD FROM XMIT & RCVR BUFFERS
ARE THE TWO WORDS EQUAL ?
NO
CRCBAD
SUBROUTINE
YES
COUNT = 0 ?
NO
INCREMENT RCVR AND XMIT BUFFER ADDRESSES
YES
MAKE TRANSMIT RING AVAILABLE FOR NEXT MESSAGE. UPDATE TRANSMIT RING POINTER ON RING MANAGEMENT STACK
END

Figure 35. Loopback Service Routine [LANCE. LEXCEPT. RVCR. LOOPBACK]

7.3.4.1 LOOPBACK HANDLING ROUTINE

The first step in this routine is to check the received message length, which must be equal to the message sent, plus four bytes if transmit CRC is enabled. This is done by checking MCNT against the transmit message word (MLENGTH). The transmit message word length is in the ring management area (see Ring Management). The size of the transmit message is moved into MLENGTH when loopback occurs during the diagnostics.

In normal operation with the application breadboard, an arbitrary message size can be written into DLENGTH, followed by pushing the message interrupt button, and a message of that data size is transmitted.

If the message sizes do not match, an error subroutine is called. Next the packet check is made. In loopback mode, transmit data chaining cannot occur and the maximum received message size is 36 bytes. The receive buffers are larger than 36 bytes and since there must be only one packet per message, the start- and end-of-packet bits must be set. If both are not set, an error routine is called.

After the packet check, the receive and transmit buffer starting addresses are retrieved. The receive buffer address is taken from the receive descriptor, while the transmit buffer address is taken out of LOOPXADD from the Ring Management Area. LOOPXADD is moved into the Ring Management Area during the transmit interrupt routine. By using these addresses, the transmit and receive message data can be compared to verify proper transmission and reception. If any part of the message has been altered, an error subroutine is called.

When the entire message has been compared, the program checks if it is in the diagnostic mode. As stated previously, a software CRC is generated for the test message, and is compared to LANCE's hardware CRC. If any of the two do not match, an error subroutine is called. This comparison is only performed during diagnostics. If the program is in loopback, but not in diagnostics, this part of the routine is omitted.

After all comparisons are made, the transmit descriptor pointer is updated. The transmit descriptor pointer is not updated in the transmit interrupt routine because the descriptor points to the transmit buffer needed for the loopback data comparison. If the transmit buffer is made available before the data comparison, the buffer might be written over, thus invalidating the comparison.

7.3.4.2 RECEIVE INTERRUPT NORMAL OPERATION

If no errors are detected, and the LANCE is not in loopback, the receive interrupt routine proceeds normally. A packet check then searches for any errors with the start-of-packet and end-of-packet bits (see Figure 36). The ring management status register contains information on the status of these bits. A receive message coming in with the start-of-packet bit set and the end-of-packet bit not set indicates that more than one buffer contains the message. In this situation, the start-of-packet bit is set in the ring management status register. If the following descriptor contains another start-of-packet bit, an error flag is set since an end-of-packet has not been detected between the two start-of-packets. If the end-of-packet bit is set in the descriptor, the whole packet is received and the status register is set accordingly. The same holds true for two end-of-packets detected sequentially. If two end-of-packets are detected, without a start-of-packet in between them, an error flag is raised. These error flags call application-dependent subroutines to service the error.

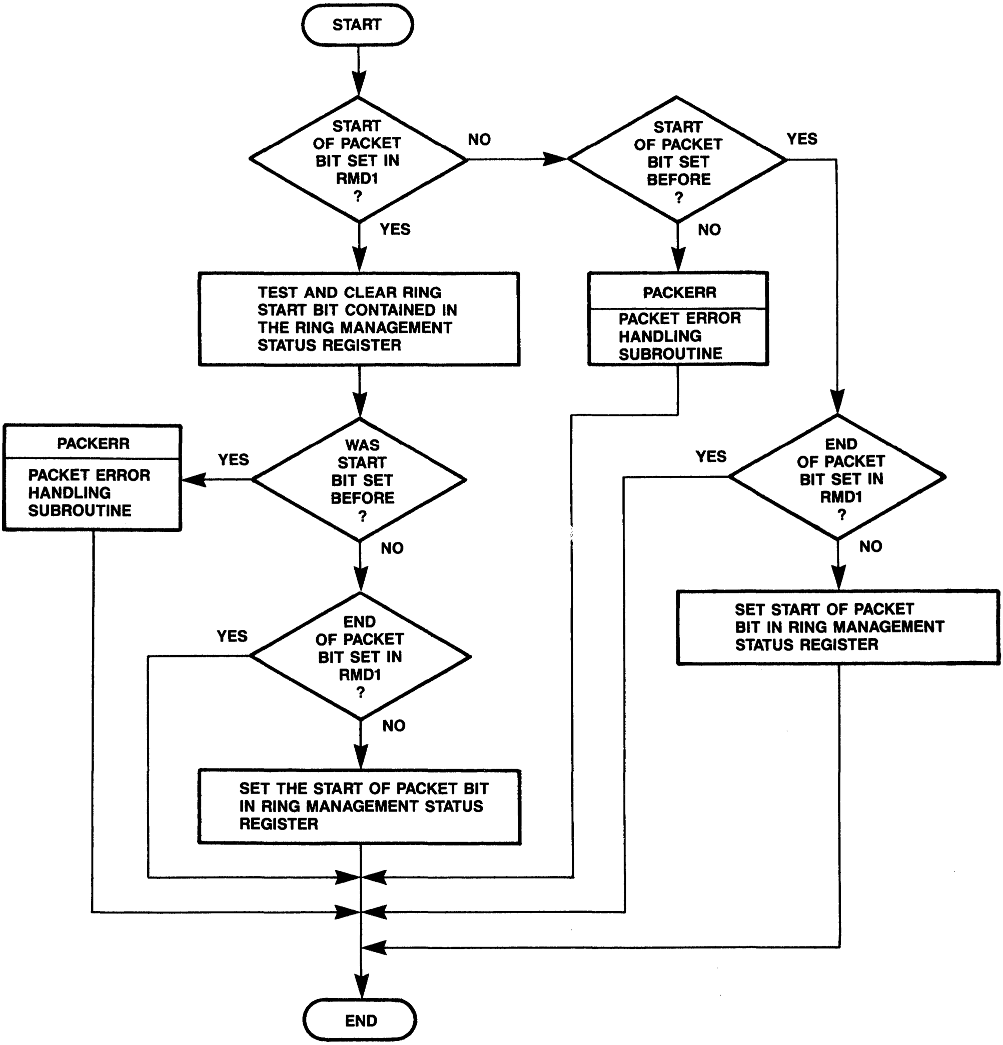

START
START OF PACKET BIT SET IN RMD1 ?
NO
YES
START OF PACKET BIT SET BEFORE ?
YES
NO
TEST AND CLEAR RING START BIT CONTAINED IN THE RING MANAGEMENT STATUS REGISTER
PACKERR
PACKET ERROR HANDLING SUBROUTINE
PACKERR
PACKET ERROR HANDLING SUBROUTINE
YES
WAS START BIT SET BEFORE ?
NO
YES
END OF PACKET BIT SET IN RMD1 ?
NO
SET START OF PACKET BIT IN RING MANAGEMENT STATUS REGISTER
YES
END OF PACKET BIT SET IN RMD1 ?
NO
SET THE START OF PACKET BIT IN RING MANAGEMENT STATUS REGISTER
END

Figure 36. Packet Status Discrepancy Check Routine [LANCE. LEXCEPT. RCVR. PACKCHECK]

Once the packet check is complete, the received message is moved to an output device or to another memory location. Once moved, this layer of software no longer handles the message, instead the upper level software takes over. The upper level reformats the message and combines messages by sequence number as well as routing them to their final destination.

In this Application Note, the message is simply moved to another memory block. This can be accomplished effectively by changing the receive buffer starting address in the receive message descriptor to the address of an empty memory block. Then the used buffer address can be passed on to the next software layer. The same holds for transmitted messages. If the number of messages exceeds the number of available buffers, the host either rejects the last message and waits for a free buffer or moves the message into an empty memory block. When a descriptor is free, the host then changes the transmit buffer address in the descriptor.

The message is moved from the message buffer into the output memory a word at a time. Each time a word is moved, a count of the message length is decremented and checked to see if any messages still need to be transferred. Once the entire message is out of the buffer, the ring management area is updated by moving the last receive descriptor pointer to the next descriptor to be used by the LANCE. The LANCE again owns the buffer.

The start bit in the ring management area undergoes a final check. A set bit indicates that this receive buffer was only one of several buffers needed for the full message. If more buffers are associated with this message, a jump back to the beginning of the receive interrupt routine is made and the routine is executed again until the entire message is processed. If the start bit is not set, this was the only buffer needed for the message, thus, the routine is completed.

7.3.5 INITIALIZATION DONE IN1ERRUPT HANDLING SUBMODULE

The final submodule of the LANCE interrupt processing routine, shown in Figure 37, is the Initialization Done Interrupt Handling Submodule. This submodule determines if the host was interrupted because the LANCE had just completed initialization. CSR0 is checked to see if the IDON bit is set. If so, a flag-raising routine is called.

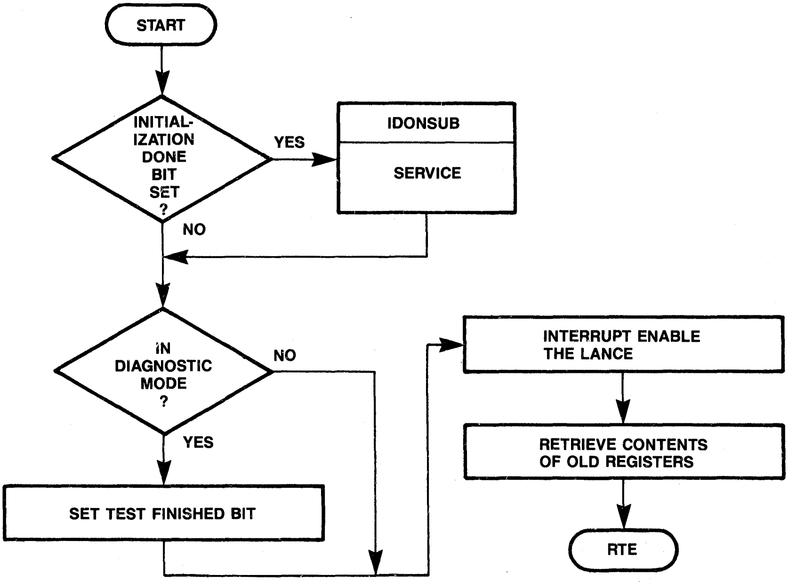

START
INITIAL-IZATION DONE BIT SET ?
YES
NO
IDONSUB
SERVICE
IN DIAGNOSTIC MODE ?
NO
YES
SET TEST FINISHED BIT
INTERRUPT ENABLE THE LANCE
RETRIEVE CONTENTS OF OLD REGISTERS
RTE

Figure 37. Initialization Done Interrupt Handling Submodule [LANCE. LEXCEPT. IDON]

The diagnostic bit in the ring management register is then checked. If the LANCE is in the diagnostic routine, the test done bit is set. The test done bit indicates a message has been loopbacked and checked and the next portion of the diagnostics may begin. This bit is checked after both a completed interrupt service routine and a Return From Exception.

At this point, all of the LANCE's interrupting conditions have been checked and serviced. During the exception processing, the LANCE's interrupting capabilities are disabled because another interrupt from the LANCE should not be performed while the first one is being serviced. Since servicing is complete the Interrupt Enable bit can again be set again in CSR0.

The module is concluded by moving the contents of the old registers off the stack and back into the register locations. A Return From Exception is then executed.

7.4 MESSAGE INTERRUPT SOFTWARE MODULE

The Message Interrupt Software Module, shown in Figure 38, is the routine that services the "push-button" message generator used in Aplication Note's hypothetical design. This push button is used for demonstration and debugging. The software description is included in this Application Note because the service routine is general enough to be adapted to any type of interrupt. The source of the interrupt may be anything, such as another processor controlling a terminal, a file server, or, as in this case, a push-button switch.

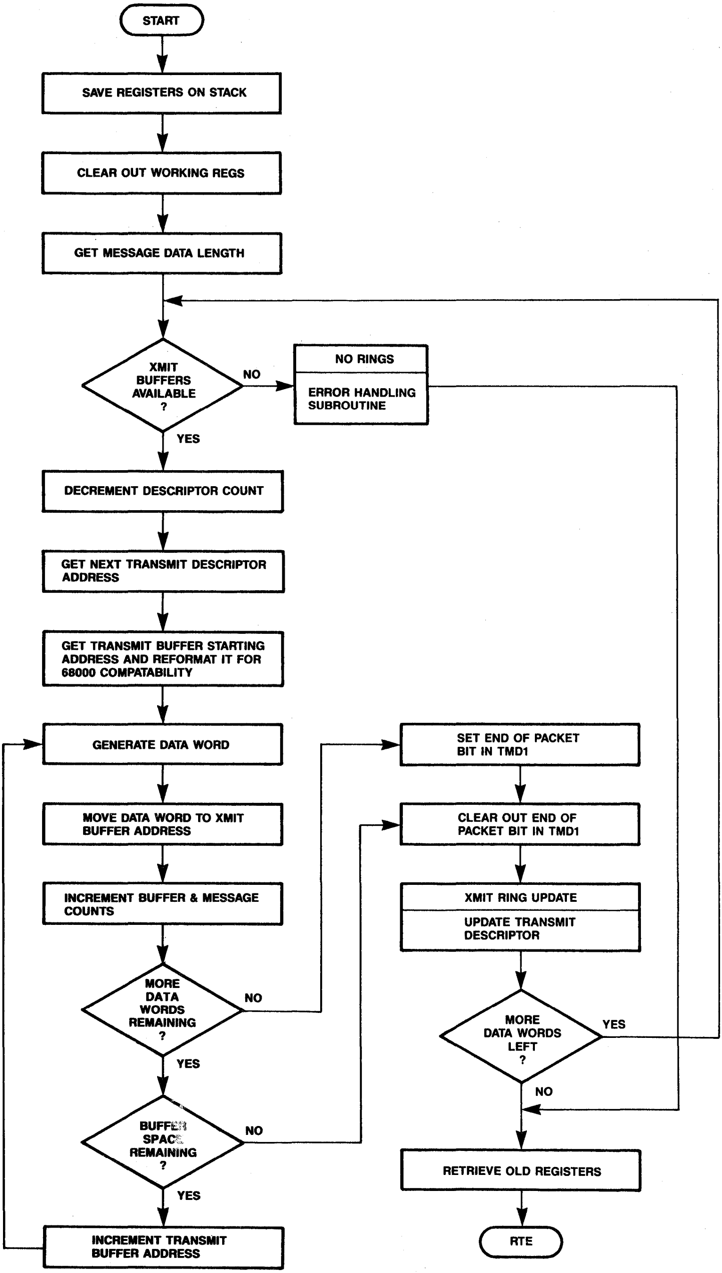

START
SAVE REGISTERS ON STACK
CLEAR OUT WORKING REGS
GET MESSAGE DATA LENGTH
XMIT BUFFERS AVAILABLE ?
NO
YES
NO RINGS
ERROR HANDLING SUBROUTINE
DECREMENT DESCRIPTOR COUNT
GET NEXT TRANSMIT DESCRIPTOR ADDRESS
GET TRANSMIT BUFFER STARTING ADDRESS AND REFORMAT IT FOR 68000 COMPATABILITY
GENERATE DATA WORD
MOVE DATA WORD TO XMIT BUFFER ADDRESS
INCREMENT BUFFER & MESSAGE COUNTS
MORE DATA WORDS REMAINING ?
NO
YES
BUFFER SPACE REMAINING ?
NO
YES
INCREMENT TRANSMIT BUFFER ADDRESS
SET END OF PACKET BIT IN TMD1
CLEAR OUT END OF PACKET BIT IN TMD1
XMIT RING UPDATE
UPDATE TRANSMIT DESCRIPTOR
MORE DATA WORDS LEFT ?
YES
NO
RETRIEVE OLD REGISTERS
RTE

Figure 38. Message Interrupt Software Module [LANCE. MESSAGE]

There is, however, one major difference between this module and one used in real life. In addition to servicing the transmit message, this module also generates the message.

This module's basic function is to fill up as many transmit buffers as needed for the outgoing message, and to set the status information in the message descriptor accordingly.

The message interrupt is hardware set at interrupt level 5. Upon interrupt, the processor autovectors to memory location $0B00, the location of this module.

The software first saves the contents of the old registers and then clears the registers for use. Next, the message data length is retrieved from DLENGTH in the Ring Management Area. This message data length is manually written into before the message button is depressed.

Next, the Descriptor Count is checked for available transmit buffers. If none are available, an error subroutine is called, which sets an error bit in the status area, and the routine is over. If a buffer is available, the descriptor count is decremented and the Next Descriptor Ring address is moved into a register from the Ring Management Area. The Next Transmit Descriptor points to the descriptor of the next transmit buffer. The transmit buffer address is then moved in from the descriptor and reformatted for MK68000 compatability.

A message data word is then generated. The message consists of byte values ranging from zero to the hex value specified by DLENGTH. In other words, if the value $20 is written into DLENGTH and a message is generated, it consists of 32 bytes. The value of the first byte is $00, the second, $01, the third, $03, the 32nd, $1F, and so forth.

Each time a data word is generated, the transmit buffer address is incremented along with the buffer and message counts. The buffer count records how many bytes have been moved into the present transmit buffer. If the transmit message length exceeds the buffer size, more then one transmit buffer is used. The message count keeps track of the number of generated message words. If this amount equals DLENGTH, the message generation part of this module is complete.

If the transmit buffer is full, and more data words need to be generated, the transmit descriptor for that buffer is updated, and the rest of the message is placed in the next available transmit message buffer. In this case, the first transmit descriptor has the start-of-packet bit set, but the end-of-packet bit is not set. Figure 39 shows a flow chart of the transmit descriptor update submodule.

Once the entire message has been moved into the message buffer, or buffers, the last transmit descriptor is updated with the status information, along with a 2's complement of the number of bytes placed in its buffer. Finally, the old registers are retrieved, and a Return From Exception is executed.

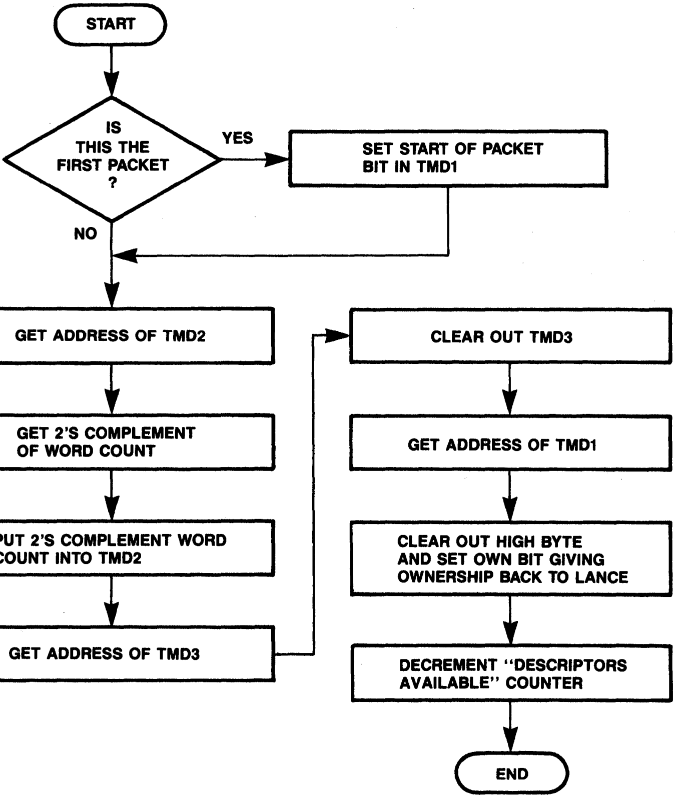

START
IS THIS THE FIRST PACKET ?
YES
SET START OF PACKET BIT IN TMD1
NO
GET ADDRESS OF TMD2
GET 2'S COMPLEMENT OF WORD COUNT
PUT 2'S COMPLEMENT WORD COUNT INTO TMD2
GET ADDRESS OF TMD3
CLEAR OUT TMD3
GET ADDRESS OF TMD1
CLEAR OUT HIGH BYTE AND SET OWN BIT GIVING OWNERSHIP BACK TO LANCE
DECREMENT "DESCRIPTORS AVAILABLE" COUNTER
END

Figure 39. Transmit Ring Update Submodule [MESSAGE. XMIT RING UPDATE]

8.0 APPENDICES

Appendix A (paragraph 8.1) presents the initialization assembly code. Appendix B (paragraph 8.2) provides the LANCE interrupt assembly code. Appendix C (paragraph 8.3) addresses the message interrupt assembly code.

8.1 APPENDIX A, INITIALIZATION ASSEMBLY CODE

The following pages provide a listing of the initialization assembly code.

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68

Line S Location  Value          Source
   1 0
   2 0
   3 0
   4 0
   5 0                          *******************************************************************+
   6 0                          *+****************************************************************+*
   7 0                          **                                                                **
   8 0                          ** MODULE NAME:  INITIAL.                                         **
   9 0                          **                                                                **
  10 0                          ** AUTHOR:  JIM FONTAINE                                          **
  11 0                          **                                                                **
  12 0                          ** PROGRAM: LANCE                                                 **
  13 0                          **                                                                **
  14 0                          ** LATEST REVISION DATE:  JANUARY 20,1984                         **
  15 0                          **                                                                **
  16 0                          ********************************************************************
  17 0                          **                                                                **
  18 0                          ** DESCRIPTION:  THIS MODULE WILL INITIALIZE THE ETHERNET NODE    **
  19 0                          ** AND MAKE IT OPERATIONAL.  THIS MODULE IS COMPOSED OF SIX       **
  20 0                          ** SUBMODULES.  THEY ARE: CLEAR, BLKMOVE, RRINGINIT, TRINGINIT,   **
  21 0                          ** DIAG, AND NORM.                                                **
  22 0                          ** THE CLEAR SUBMODULE CLEARS OUT THE WORKING REGISTERS, RESETS   **
  23 0                          ** THE SYSTEM,  DEFINES THE INTERRUPT VECTORS AND STACK           **
  24 0                          ** LOCATIONS. THE BLKMOVE SUBMODULE MOVES THE INITIALIZATION      **
  25 0                          ** BLOCK FROM PROGRAM MEMORY INTO THE MEMORY SPACE ALLOCATED FOR **
  26 0                          ** LANCE'S INITIALIZATION BLOCK. RRINGINIT SUBMODULE INITIALIZES **
  27 0                          ** THE RECEIVE MESSAGE DESCRIPTOR RINGS. THE TRINGINIT SUBMODULE **
  28 0                          ** INITIALIZES THE TRANSMIT MESSAGE DESCRIPTOR RINGS.  THE        **
  29 0                          ** DIAGNOSTIC SUBMODULE RUNS THROUGH A INTERNAL AND EXTERNAL      **
  30 0                          ** LOOPBACK ROUTINE TO TEST THE PROPER OPERATION OF THE LANCE     **
  31 0                          ** AND ADDITIONAL HARDWARE.  FINALLY, THE NORMAL SUBMODULE        **
  32 0                          ** INITIALIZES THE LANCE IN A NORMAL MODE OF OPERATION.           **
  33 0                          **                                                                **
  34 0                          ******************************************************************+*
  35 0                          *******************************************************************+
  36 0
  37 0
  38 0                          **************************************************
  39 0                          *                                                *
  40 0                          *               EQUATE TABLE                     *
  41 0                          *                                                *
  42 0                          **************************************************
  43 0
  44 0                                   XREF     DATINTR
  45 0                                   XREF     LANINTR
  46 0
  47 0                          *   DESCRIPTOR RING STACK ALLOCATION   *
  48 0
  49 0
  50 0          00000000        RLEN     EQU      $00                    ;38C0   RECEIVER RING LENGTH
  51 0          00000002        TLEN     EQU      $02                    ;38C2   TRANSMIT RING LENGTH
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68

Line S Location  Value          Source
  52 0           00000004       RRNGBASE EQU      $04              ;38C4   RECEIVER RING BASE ADDRESS
  53 0           00000008       TRNGBASE EQU      $08              ;38C8   TRANSMIT RING BASE ADDRESS
  54 0           0000000C       RRNGBOT  EQU      $0C              ;38CC   RECEIVER RING BOTTOM ADDRESS
  55 0           00000010       TRNGBOT  EQU      $10              ;38D0   TRANSMIT RING BOTTOM ADDRESS
  56 0           00000014       LASTRRNG EQU      $14              ;38D4   ADDRESS OF LAST RECEIVE RING USED
  57 0           00000018       LASTTRNG EQU      $18              ;38D8   ADDRESS OF LAST TRANSMIT RING USED
  58 0           0000001C       NEXTTRNG EQU      $1C              ;38DC   ADDRESS OF NEXT AVAILABLE TRANSMIT RING
  59 0           00000020       LOOPMCNT EQU      $20              ;38E0   MESSAGE COUNT OF LOOP MESSAGE
  60 0           00000022       TRINGCNT EQU      $22              ;3EE2   NUMBER OF TRANSMIT RINGS AVAILABLE TO USE
  61 0           00000024       RINGSTAT EQU      $24              ;38E4   RING STATUS
  62 0           00000028       LOOPXADD EQU      $28              ;38E8   ADDRESS OF LOOPBACK BUFFER
  63 0           0000002C       MORE     EQU      $2C              ;38EC   RUNING COUNT OF "MORE" FLAGS
  64 0           0000002E       ONE      EQU      $2E              ;38EE   RUNNING COUNT OF "ONE" FLAGS
  65 0           00000034       DLENGTH  EQU      $34              ;38F0   DESIRED MESSAGE DATA SIZE
  66 0           00000038       MLENGTH  EQU      $38              ;38F4   TOTAL LENGTH OF TRANSMITTED MESSAGE
  67 0
  68 0           00000100       BUFLEN   EQU      $0100            ;TRANSMIT & RECEIVE BUFFER LENGTH
  69 0           00000060       RBUFDIS  EQU      $0060            ;RECEIVER BUFFER DISPLACEMENT FROM DESCRIPT. RING
  70 0           00001020       TBUFDIS  EQU      $1020            ;XMIT BUFFER DISPLACEMENT FROM THE DESCRIPT RING
  71 0
  72 0
  73 0
  74 0                          *   MEMORY MAP LOCATIONS   *
  75 0
  76 0
  77 0           00002000       IADR     EQU      $2000            ;LANCES INITIALIZATION BLOCK BASE ADDRESS
  78 0           0000387C       STACK    EQU      $387C            ;STACK BASE ADDRESS
  79 0           000038C0       RINGSTK  EQU      $38C0            ;XMIT & RVCR USERS STACK BASE ADDRESS
  80 0
  81 0           00000044       MODE     EQU      $0044            ;ETHERNET MODE
  82 0           00000004       PADR1    EQU      $0004            ;PHYSICAL ADDRESS <00:15>
  83 0           00000000       PADR2    EQU      $0000            ;PHYSICAL ADDRESS <16:31>
  84 0           00000000       PADR3    EQU      $0000            ;PHYSICAL ADDRESS <32:47>
  85 0           00000008       LADR1    EQU      $0008            ;LOGICAL ADDRESS <00:15>
  86 0           00000000       LADR2    EQU      $0000            ;LOGICAL ADDRESS <16:31>
  87 0           00000000       LADR3    EQU      $0000            ;LOGICAL ADDRESS <32:47>
  88 0           00000000       LADR4    EQU      $0000            ;LOGICAL ADDRESS <48:63>
  89 0           00002018       RDRA1    EQU      $2018            ;RECEIVE DESCRIPTOR ADDRESS <00:15>
  90 0           00006000       RDRA2    EQU      $6000            ;RECEIVE DESCRIPTOR ADDRESS <16:23>
  91 0           00002058       TDRA1    EQU      $2058            ;TRANSMIT DESCRIPTOR ADDRESS <00:15>
  92 0           00004000       TDRA2    EQU      $4000            ;TRANSMIT DESCRIPTOR ADDRESS <16:23>
  93 0
  94 0           00004002       RAP      EQU      $4002            ;LANCE'S REGISTER ADDRESS PORT ADDRESS
  95 0           00004000       RDP      EQU      $4000            ;LANCE'S REGISTER DATA PORT ADDRESS
  96 0
  97 0
  98 0                          *   KEYWORD VALUE EQUATES   *
  99 0
 100 0
 101 0           00000000       CSR0     EQU      $00              ;ADDRESS OF CONTROL-STATUS REGISTER 0
 102 0           00000001       CSR1     EQU      $01              ;ADDRESS OF CONTROL-STATUS REGISTER 1
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68


Line S Location  Value          Source
 103 0          00000002        CSR2     EQU      $02                   ;ADDRESS OF CONTROL-STATUS REGISTER 2
 104 0          00000003        CSR3     EQU      $03                   ;ADDRESS OF CONTROL-STATUS REGISTER 3
 105 0
 106 0                          *   WORD VALUE DEFINITION    *
 107 0
 108 0
 109 0                                   XDEF     RAP
 110 0                                   XDEF     RDP
 111 0
 112 0                                   XDEF     CSR0
 113 0                                   XDEF     CSR1
 114 0                                   XDEF     CSR2
 115 0                                   XDEF     CSR3
 116 0
 117 0                                   XDEF     RLEN
 118 0                                   XDEF     TLEN
 119 0                                   XDEF     RRNGBASE
 120 0                                   XDEF     TRNGBASE
 121 0                                   XDEF     RRNGBOT
 122 0                                   XDEF     TRNGBOT
 123 0                                   XDEF     LASTRRNG
 124 0                                   XDEF     LASTTRNG
 125 0                                   XDEF     NEXTTRNG
 126 0                                   XDEF     LOOPMCNT
 127 0                                   XDEF     TRINGCNT
 128 0                                   XDEF     RINGSTAT
 129 0                                   XDEF     MORE
 130 0                                   XDEF     ONE
 131 0                                   XDEF     LOOPXADD
 132 0                                   XDEF     DLENGTH
 133 0                                   XDEF     TBUFLEN
 134 0                                   XDEF     MLENGTH
 135 0
 136 0                                   XDEF     DIAG
 137 0
 138            00000400                 ORG      $400
 139
 140                            **************** MAIN PROGRAM **********************
 141
 142
 143                            ****************************************************
 144                            *                                                  *
 145                            *      INITIAL.CLEAR                               *
 146                            *      INITIALIZE MK68000 AND CLEAR REGISTERS      *
 147                            *                                                  *
 148                            ****************************************************
 149
 150
 151
 152   00000400 46FC2700        CLEAR    MOVE.W   #$2700,SR             SET UP STATUS REGISTER
 153   00000404 4E70                     RESET                          RESET THE 68000
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68


Line S Location  Value          Source
 154
 155                            ***   CLEAR  ALL  DATA  REGISTERS   ***
 156
 157    00000406 4280                    CLR.L     D0                   CLEAR OUT DATA REG D0
 158    00000408 4281                    CLR.L     D1                   CLEAR OUT DATA REG D1
 159    0000040A 4282                    CLR.L     D2                   CLEAR OUT DATA REG D2
 160    0000040C 4283                    CLR.L     D3                   CLEAR OUT DATA REG D3
 161    0000040E 4284                    CLR.L     D4                   CLEAR OUT DATA REG D4
 162    00000410 4285                    CLR.L     D5                   CLEAR OUT DATA REG D5
 163    00000412 4286                    CLR.L     D6                   CLEAR OUT DATA REG D6
 164    00000414 4287                    CLR.L     D7                   CLEAR OUT DATA REG D7
 165
 166                            ***   CLEAR  ADDRESS  REGISTERS  A0 AND A4   ***
 167
 168    00000416 2040                    MOVE.L    D0,A0                CLEAR ADDRESS REG A0
 169    00000418 2240                    MOVE.L    D0,A1                CLEAR ADDRESS REG A1
 170    0000041A 2440                    MOVE.L    D0,A2                CLEAR ADDRESS REG A2
 171    0000041C 2640                    MOVE.L    D0,A3                CLEAR ADDRESS REG A3
 172    0000041E 2840                    MOVE.L    D0,A4                CLEAR ADDRESS REG A4
 173
 174                            ***   DEFINE STACK AND INTERRUPT VECTOR LOCATIONS   ***
 175
 176
 177    00000420 207C00000078            MOVE.L    #$78,A0              DEFINE VECTOR FOR "LANCE" INTERRUPT
 178    00000426 20BCFFFFFFFF            MOVE.L    #LANINTR,(A0)        MOVE VECTOR LOCATION TO LOCATION $78
 179    0000042C 207C00000074            MOVE.L    #$74,A0              DEFINE VECTOR FOR "DATA FOR XMIT" INTERRUPT
 180    00000432 20BCFFFFFFFF            MOVE.L    #DATINTR,(A0)        MOVE VECTOR LOCATION TO LOCATION $74
 181    00000438 287C000038C0            MOVE.L    #RINGSTK,A4          MOVE RING STACK LOCATION TO REGISTER A4
 182    0000043E 207C0000387C            MOVE.L    #STACK,A0            MOVE STACK LOCATION TO A0
 183    00000444 4E60                    MOVE.L    A0,USP               MOVE STACK LOCATION TO USER STACK POINTER
 184    00000446 46FCA400                MOVE.W    #$A400,SR            SET UP MASK IN STATUS REGISTER
 185    0000044A 29400028                MOVE.L    D0,LOOPXADD(A4)      CLEAR OUT LOOP ADDRESS ON STACK
 186    0000044E 19400024                MOVE.B    D0,RINGSTAT(A4)      CLEAR OUT RING STATUS
 187    00000452 3940002E                MOVE.W    D0,ONE(A4)           CLEAR OUT THE ONE'S COUNTER
 188    00000456 3940002C                MOVE.W    D0,MORE(A4)          CLEAR OUT THE MORE'S COUNTER
 189    0000045A 39400020                MOVE.W    D0,LOOPMCNT(A4)      CLEAR OUT THE MESSAGE COUNT
 190
 191                            *******************************************************
 192                            *                                                     *
 193                            *      INITIAL.BLKMOVE                                *
 194                            *      MOVE INITIALIZATION BLOCK                      *
 195                            *                                                     *
 196                            *******************************************************
 197
 198
 199    0000045E 207C0000071C            MOVE.L    #PROGIADR,A0         GET STARTING ADDRESS OF PROGRAM INIT. BLOCK
 200    00000464 227C00002000            MOVE.L    #IADR,A1             GET STARTING ADDRESS OF LANCE INIT. BLOCK
 201
 202    0000046A 2290           RETURN   MOVE.L    (A0),(A1)            MOVE LONG WORD OF PROG BLOCK INTO LANCE BLOCK
 203    0000046C 0C04000F                CMPI.B    #15,D4               SEE IF WE ARE UP TO ADDRESS IADR <23:00>+20 (OCT)
 204    00000470 6E00000A                BGT       COUNT16              IF SO BRANCH TO COUNT = 16
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68


Line S Location  Value         Source
 205   00000474 5848           INCREM   ADD      #04,A0           INCREMENT THE PROGRAM INIT BLOCK ADDRESS BY 4
 206   00000476 5849                    ADD      #04,A1           INCREMENT THE LANCE INIT BLOCK ADDRESS IN A1 BY 4
 207   00000478 5844                    ADD      #04,D4           INCREMENT COUNTER BY 4
 208   0000047A 60EE                    BRA      RETURN           GO BACK AND LOAD THE NEXT LONG WORD
 209   0000047C 0C040010       COUNT16  CMPI.B   #16,D4           SEE IF THE COUNT IS EQUAL TO 16
 210   00000480 66000030                BNE      COUNT20          IF NOT BRANCH TO COUNT = 20
 211
 212                           ***   RECEIVE RING LENGTH CALCULATION   ***
 213
 214   00000484 2010                    MOVE.L   (A0),D0          MOVE RECEIVE RING ADDRESS AND LENGTH TO D0
 215   00000486 E758                    ROL.W    #3,D0            MOVE RECEIVE LENGTH TO LOWER BYTE OF LONG WORD
 216   00000488 1200                    MOVE.B   D0,D1            SAVE A COPY OF CODED RECEIVER RING LENGTH IN REG D1
 217   0000048A 02010007                ANDI.B   #$07,D1          CLEAR OUT EVERYTHING EXCEPT FOR THE LENGTH
 218   0000048E 7A01                    MOVEQ    #01,D5           LOAD REG D1 WITH  1 FOR CALC. RING LENGTH
 219   00000490 04010001       NXTPOWER SUBI.B   #01,D1           DECREMENT LENGTH COUNT
 220   00000494 6D06                    BLT.S    CALCDONE         IF COUNTER IS LESS THAT "0" CALCULATION IS FINISHED
 221   00000496 CAFC0002                MULU     #2,D5            MULTIPLY BY 2 TO CALCULATE POWER
 222   0000049A 60F4                    BRA.S    NXTPOWER         CALCULATE NEXT POWER OF 2
 223
 224   0000049C 3885           CALCDONE MOVE.W   D5,RLEN(A4)      MOVE RECEIVER LENGTH OUT ONTO THE RING STACK
 225   0000049E E658                    ROR.W    #3,D0            MOVE THE WORD BACK INTO ORIGINAL POSISION
 226   000004A0 024000FF                ANDI.W   #$00FF,D0        CLEAR OUT THE STATUS FOR THE ADDRESS WORD
 227   000004A4 4840                    SWAP     D0               REFORMAT THE WORDS FOR 68000 COMPATABILITY
 228   000004A6 29400004                MOVE.L   D0,RRNGBASE(A4)  PUT RECEIVE DESCRIPTOR BASE ADDRESS ON RING STACK
 229   000004AA 29400014                MOVE.L   D0,LASTRRNG(A4)  PUT RVCR DESCRIPTOR BASE ADDRESS IN LAST RVCR RING SPOT
 230   000004AE 2A40                    MOVE.L   D0,A5            SAVE A COPY OF THE RVCR RING BASE ADD. IN REG A5
 231   000004B0 60C2                    BRA      INCREM           GO INCREMENT THE REGISTERS
 232
 233                           ***   TRANSMIT RING LENGTH CALCULATION    ***
 234
 235   000004B2 2010           COUNT20  MOVE.L   (A0),D0          MOVE TRANSMIT RING ADDRESS AND LENGTH TO D0
 236   000004B4 E758                    ROL.W    #3,D0            MOVE XMIT LENGTH TO LOWER BYTE OF LONG WORD
 237   000004B6 1200                    MOVE.B   D0,D1            MAKE A COPY OF CODED XMIT RING LENGTH IN REG D1
 238   000004B8 02010007                ANDI.B   #$07,D1          CLEAR OUT EVERYTHING EXCEPT FOR THE LENGTH
 239
 240   000004BC 7C01                    MOVEQ    #01,D6           LOAD REG D1 WITH 1 FOR CALC. RING LENGTH
 241   000004BE 04010001       NXPOWER  SUBI.B   #01,D1           DECREMENT LENGTH COUNT
 242   000004C2 6D06                    BLT.S    CALCDNE          IF COUNTER IS LESS THAT "0" CALCULATION IS FINISHED
 243   000004C4 CCFC0002                MULU     #2,D6            MULTIPLY BY 2 TO CALCULATE POWER
 244   000004C8 60F4                    BRA.S    NXPOWER          CALCULATE NEXT POWER OF 2
 245
 246   000004CA 39460002       CALCDNE  MOVE.W   D6,TLEN(A4)      MOVE XMIT LENGTH OUT ONTO THE RING STACK
 247   000004CE 39460022                MOVE.W   D6,TRINGCNT(A4)  MOVE XMIT RING COUNT OUT TO "COUNTER STACK LOCATION
 248   000004D2 E658                    ROR.W    #3,D0            MOVE THE WORD BACK INTO ORIGINAL POSITION
 249   000004D4 024000FF                ANDI.W   #$00FF,D0        CLEAR THE STATUS OUT OF THE ADDRESS WORD
 250   000004D8 4840                    SWAP     D0               REFORMAT THE WORDS FOR 68000 COMPATABILITY
 251   000004DA 29400008                MOVE.L   D0,TRNGBASE(A4)  PUT XMIT DESCRIPTOR BASE ADDRESS ON RING STACK
 252   000004DE 2940001C                MOVE.L   D0,NEXTTRNG(A4)  PUT XMIT DESCRIPTOR BASE ADD IN NEXT RING SPOT
 253   000004E2 29400018                MOVE.L   D0,LASTTRNG(A4)  PUT A COPY OF XMIT DESCR. ADD. IN LAST RING SPOT
 254   000004E6 2C40                    MOVE.L   D0,A6            SAVE A COPY OF THE BASE ADDRESS IN REG A6
 255
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68

Line S Location  Value          Source
 256
 257                            *********************************************************
 258                            *                                                       *
 259                            *    INITIAL.RRINGINIT                                  *
 260                            *    INITIALIZE THE RECEIVE MESSAGE DESCRIPTOR RINGS    *
 261                            *                                                       *
 262                            *********************************************************
 263
 264                            ***   GENERATE RMD0, AND RMD1   ***
 265
 266   000004E8 220D                     MOVE.L   A5,D1                 GET COPY OF RVCR RING BASE ADD. AND PUT IT IN D1
 267   000004EA 068100000060             ADD.L    #RBUFDIS,D1           GENERATE RECEIVE BUFFER ADDRESS BY ADDING DISPLACEMENT
 268   000004F0 2401                     MOVE.L   D1,D2                 COPY
 269   000004F2 600A                     BRA.S    FIRSTRCV              GO INITIALIZE FIRST RECEIVE RING
 270
 271   000004F4 588D            NEXTRRNG ADDQ.L   #$04,A5               GENERATE NEXT ADDRESS OF MESSAGE DESCRIPTOR
 272   000004F6 068200000100             ADDI.L   #BUFLEN,D2            INCREMENT THE BUFFER DISPLACEMENT WITH BUFFER LENGTH
 273   000004FC 2202                     MOVE.L   D2,D1
 274   000004FE 008180000000    FIRSTRCV ORI.L    #$80000000,D1         SET OWN BIT IN DESCRIPTOR WORD
 275   00000504 4841                     SWAP     D1                    MAKE WORDS COMPATABILITY WITH 68000 FORMAT
 276   00000506 2A81                     MOVE.L   D1,(A5)               MOVE LONG WORD INTO DESCRIPTOR RING ADDRESS
 277
 278                            ***   GENERATE RMD2, AND RMD3   ***
 279
 280   00000508 588D                     ADDQ.L   #$04,A5               GENERATE NEXT ADDRESS OF DESCRIPTOR WORD
 281   0000050A 203C00000100             MOVE.L   #BUFLEN,D0            GET RING BUFFER LENGTH
 282   00000510 0A40FFFF                 EORI.W   #$FFFF,D0             GET 2'S COMPLEMENT BY EXCLUSIVE ORING AND,
 283   00000514 06400001                 ADDI.W   #$01,D0               ADDING 1 TO THE RESULT. THIS GIVES BYTE COUNT
 284   00000518 4840                     SWAP     D0                    MAKE WORDS CAMPATABILITY WITH 68000 FORMAT
 285   0000051A 2A80                     MOVE.L   D0,(A5)               MOVE LONG WORD TO MESSAGE DESCRIPTOR 2 AND 3
 286
 287                            ***   GENERATE NEXT RECEIVE DESCRIPTOR RING   ***
 288
 289   0000051C 04050001                 SUBI.B   #$01,D5               DECREMENT THE RECEIVER RING COUNT
 290   00000520 66D2                     BNE.S    NEXTRRNG              IF THERE IS STILL ANOTHER RING LEFT, GO INIT. IT
 291   00000522 598D                     SUBQ.L   #$04,A5               OR, POINT BACK TO THE BEGINING OF THE ADDRESS
 292   00000524 294D000C                 MOVE.L   A5,RRNGBOT(A4)        MOVE IT OUT ONTO THE RING STACK
 293
 294                            *********************************************************
 295                            *                                                       *
 296                            *    INITIAL.TRINGINIT                                  *
 297                            *    INITIALIZE THE TRANSMIT MESSAGE DESCRIPTOR RINGS   *
 298                            *                                                       *
 299                            *********************************************************
 300
 301                            ***   GENERATE TMD0, AND TMD1   ***
 302
 303   00000528 220E                     MOVE.L   A6,D1                 GET XMIT RING BASE ADDRESS AND PUT IT IN D1
 304   0000052A 068100001020             ADD.L    #TBUFDIS,D1           GENERATE XMIT BUFFER ADDRESS BY ADDING DISPLACEMENT
 305   00000530 2401                     MOVE.L   D1,D2                 COPY IT
 306   00000532 600A                     BRA.S    FIRSTXM               GO GENERATE FIRST TRANSMIT RING
```

```
Line S Location  Value          Source

00000534 588E          NXTTRNG  ADDQ.L   #$04,A6               GENERATE NEXT ADDRESS OF MESSAGE DESCRIPTOR
00000536 068200000100           ADDI.L   #BUFLEN,D2
0000053C 2202                   MOVE.L   D2,D1
0000053E 4841          FIRSTXM  SWAP     D1
00000540 2C81                   MOVE.L   D1,(A6)               MOVE LONG WORD INTO DESCRIPTOR RING ADDRESS

                       ****   GENERATE TMD2, AND TMD3   ***

00000542 588E                   ADDQ.L   #$04,A6               GENERATE NEXT ADDRESS OF DESCRIPTOR WORD
00000544 203C00000100           MOVE.L   #BUFLEN,D0            GET RING BUFFER LENGTH
0000054A 0A40FFFF               EORI.W   #$FFFF,D0             GET 2'S COMPLEMENT BY EXCLUSIVE ORING AND,
0000054E 06400001               ADDI.W   #$01,D0               ADDING 1 TO THE RESULT. THIS GIVES BYTE COUNT
00000552 4840                   SWAP     D0                    REFORMAT FOR 68000 COMPATABILITY
00000554 2C80                   MOVE.L   D0,(A6)               MOVE LONG WORD TO MESSAGE DESCRIPTOR 2 AND 3

                       ***   GENERATE NEXT TRANSMIT DESCRIPTOR RING   ***

00000556 04060001               SUBI.B   #$01,D6               DECREMENT THE MESSAGE RING COUNT
0000055A 66D8                   BNE.S    NXTTRNG               IF THERE IS STILL ANOTHER RING LEFT, GO INIT. IT
0000055C 598E                   SUBQ.L   #$04,A6               POINT BACK TO THE BEGINNING OF THE ADDRESS
0000055E 294E0010               MOVE.L   A6,TRNGBOT(A4)        GO MOVE IT OUT ONTO THE STACK

                       ** CALL DIAGNOSTICS ROUTINE **

00000562 4EB90000059A           JSR      DIAG                  CALL DIAGNOSTICS SUBROUTINE


                       *******************************************************
                       *                                                     *
                       *      INITIAL.NORM                                   *
                       *      INITIALIZES FOR NORMAL OPERATION               *
                       *                                                     *
                       *******************************************************

                       ***   SET UP FOR NORMAL OPERATION   ***

00000568 207C0000071C           MOVE.L   #PROGIADR,A0          GET STARTING ADDRESS OF PROGRAM INIT. BLOCK
0000056E 2290                   MOVE.L   (A0),(A1)             MOVE MODE INFORMATION INTO INIT BLOCK

                       *   LOOPBACK CHECKING   *
00000570 397C00000024           MOVE.W   #0,RINGSTAT(A4)       CLEAR OUTRING STATUS REGISTER
00000576 3010                   MOVE.W   (A0),D0               GET MODE INFORMATION AND PUT IN REG D0
00000578 08000002               BTST     #02,D0                SEE IF WE ARE IN LOOPBACK MODE
0000057C 6712                   BEQ.S    GOINIT                IF NOT GO INITIALIZE THE CSR REGISTERS
0000057E 08EC00010024           BSET.B   #01,RINGSTAT(A4)      IF SO, SET THE LOOP STATUS BIT
00000584 08000006               BTST     #06,D0                SEE IF WE ARE IN INTERNAL LOOPBACK
00000588 6706                   BEQ.S    GOINIT                IF NOT, GO INITIALIZE THE CSR REGISTERS
0000058A 08EC00000024           BSET.B   #00,RINGSTAT(A4)      IF SO, SET THE INTERNAL LOOP STATUS BIT
00000590 4EB9000006CC  GOINIT   JSR      CSRINIT               GO START UP THE LANCE

```

```
Line S Location  Value          Source
 358   00000596 4E71            WAIT     NOP                            WAIT HERE UNTIL AN INTERRUPT OCCURS
 359   00000598 60FC                     BRA      WAIT                  WAIT SOME MORE
 360
 361
 362
 363                            ******************************************************
 364                            *                                                    *
 365                            *      INITIAL.DIAG                                  *
 366                            *      LANCE DIAGNOSTIC ROUTINE                      *
 367                            *                                                    *
 368                            ******************************************************
 369
 370                            ***   DIAGNOSTIC MESSAGE GENERATION   ***
 371
 372   0000059A 4281            DIAG     CLR.L    D1                    CLEAR OUT REGISTER D1
 373   0000059C 4282                     CLR.L    D2                    CLEAR OUT REGISTER D2
 374   0000059E 227C00002000             MOVE.L   #IADR,A1              GET INITIALIZATION STARTING ADDRESS
 375   000005A4 08EC00040024             BSET.B   #04,RINGSTAT(A4)      SET DIAGNOSTIC BIT IN STATUS REG
 376
 377   000005AA 08EC00010024             BSET.B   #01,RINGSTAT(A4)      SET THE LOOP STATUS BIT
 378   000005B0 08EC00000024             BSET.B   #00,RINGSTAT(A4)      SET THE INTERNAL LOOP STATUS BIT
 379
 380   000005B6 206C001C                 MOVE.L   NEXTTRNG(A4),A0       GET XMIT RING BASE ADDRESS
 381   000005BA 2610                     MOVE.L   (A0),D3               GET XMIT BUFFER BASE ADDRESS
 382   000005BC 024300FF                 ANDI.W   #$00FF,D3             CLEAR STATUS OUT OF WORD
 383   000005C0 2A03                     MOVE.L   D3,D5                 MAKE A COPY OF IT
 384   000005C2 4843                     SWAP     D3                    MOVE WORDS FOR 68000 COMPATABILITY
 385   000005C4 2643                     MOVE.L   D3,A3                 MOVE XMIT BUFFER ADDRESS INTO ADD REG
 386   000005C6 26BCAAAAAAAA    MESS     MOVE.L   #$AAAAAAAA,(A3)       MOVE TEST MESSAGE WORD INTO BUFFER ADDRESS
 387   000005CC 5841                     ADDQ.W   #04,D1                INCREMENT MESSAGE COUNTER BY 4 (BYTES)
 388   000005CE 584B                     ADDQ.W   #04,A3                INCREMENT THE BUFFER ADDRESS
 389   000005D0 0C41001C                 CMP.W    #$1C,D1               CHECK TO SEE IF THE MESSAGE IS 28 BYTES LONG
 390   000005D4 6DF0                     BLT.S    MESS                  IF SO, GO ADD MORE TO THE MESSAGE
 391   000005D6 4EB900000706             JSR      CRCGEN                JUMP TO SOFTWARE CRC GENERATOR
 392   000005DC 397C00200038             MOVE.W   #$20,MLENGTH(A4)      MOVE A MESSAGE LENGTH OF 32 BYTES OUT TO
 393                            *                                       RING MANG. AREA FOR LATER COMPARISON
 394   000005E2 397C001C0034             MOVE.W   #$1C,DLENGTH(A4)      MOVE DATA SIZE OUT FOR LATER COMPARISON
 395
 396
 397                            ***  TEST INTERNAL LOOPBACK WITH TRANSMIT CRC ENABLED   ***
 398
 399   000005E8 00458300                 ORI.W    #$8300,D5             SET OWNERSHIP BIT IN XMIT RING
 400   000005EC 2085                     MOVE.L   D5,(A0)               MOVE TMD1 TO RING LOCATION
 401   000005EE 5888                     ADDQ.L   #04,A0                GENERATE ADDRESS OF TMD2
 402   000005F0 20BCFFE40000             MOVE.L   #$FFE40000,(A0)       MOVE BYTE COUNT(28 BYTES) TO TMD2 ON DESCRIPTOR
 403                            *                                       RING AND CLEAR OUT TMD3
 404   000005F6 5588                     SUBQ.L   #02,A0                POINT TO TMD1
 405   000005F8 32BC8044                 MOVE.W   #$8044,(A1)           SET UP INIT BLOCK TO INTERNAL LOOPBACK
 406   000005FC 4EB9000006CC             JSR      CSRINIT               GO START UP THE LANCE
 407   00000602 08AC00050024    CHECK1   BCLR.B   #05,RINGSTAT(A4)      SEE IF THIS PORTION OF THE TEST IS COMPLETE
 408   00000608 67F8                     BEQ      CHECK1                CHECK AGAIN
```

```
Line S Location  Value          Source
 409
 410                            ***   TURN OFF THE LANCE   ***
 411    0000060A 247C00004002             MOVE.L   #RAP,A2                SELECT LANCE'S REGISTER ADDRESS PORT TO WRITE TO
 412    00000610 34BC0000                 MOVE.W   #CSRO,(A2)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSRO
 413    00000614 267C00004000             MOVE.L   #RDP,A3                SELECT LANCE'S REGISTER DATA PORT TO WRITE TO
 414    0000061A 36BC00FF                 MOVE.W   #$00FF,(A3)            TURN OFF THE LANCE
 415
 416                            ***   TEST INTERNAL LOOPBACK WITH TRANSMIT CRC DISABLED   ***
 417
 418    0000061E 00458300                 ORI.W    #$8300,D5              SET OWNERSHIP BIT IN XMIT RING
 419    00000622 3085                     MOVE.W   D5,(A0)                MOVE TMD1 TO RING LOCATION
 420    00000624 5488                     ADDQ.L   #02,A0                 GENERATE ADDRESS OF TMD2
 421    00000626 20BCFFE00000             MOVE.L   #$FFE00000,(A0)        MOVE BYTE COUNT(32 BYTES) TO TMD2 ON DESCRIPTOR
 422                            *                                         RING AND CLEAR OUT TMD3
 423    0000062C 5588                     SUBQ.L   #02,A0                 POINT TO TMD1
 424    0000062E 32BC804C                 MOVE.W   #$804C,(A1)            SET UP INIT BLOCK TO INTERNAL LOOPBACK
 425    00000632 4EB9000006CC             JSR      CSRINIT                GO START UP THE LANCE
 426    00000638 08AC00050024   CHECK2    BCLR.B   #05,RINGSTAT(A4)       SEE IF THIS PORTION OF THE TEST IS COMPLETE
 427    0000063E 67F8                     BEQ      CHECK2                 CHECK AGAIN
 428
 429                            ***   TURN OFF THE LANCE   ***
 430    00000640 247C00004002             MOVE.L   #RAP,A2                SELECT LANCE'S REGISTER ADDRESS PORT TO WRITE TO
 431    00000646 34BC0000                 MOVE.W   #CSRO,(A2)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSRO
 432    0000064A 267C00004000             MOVE.L   #RDP,A3                SELECT LANCE'S REGISTER DATA PORT TO WRITE TO
 433    00000650 36BC00FF                 MOVE.W   #$00FF,(A3)            TURN OFF THE LANCE
 434
 435                            ***   TEST INTERNAL LOOPBACK WITH XMIT CRC ENABLED AND COLLISION FORCE ON ***
 436
 437    00000654 00458300                 ORI.W    #$8300,D5              SET OWNERSHIP BIT IN XMIT RING
 438    00000658 3085                     MOVE.W   D5,(A0)                MOVE TMD1 TO RING LOCATION
 439    0000065A 5488                     ADDQ.L   #02,A0                 GENERATE ADDRESS OF TMD2
 440    0000065C 20BCFFE40000             MOVE.L   #$FFE40000,(A0)        MOVE BYTE COUNT(28 BYTES) TO TMD2 ON DESCRIPTOR
 441                            *                                         RING AND CLEAR OUT TMD3
 442    00000662 5588                     SUBQ.L   #02,A0                 POINT TO TMD1
 443
 444    00000664 32BC8054                 MOVE.W   #$8054,(A1)            SET UP INIT BLOCK TO INTERNAL LOOPBACK
 445    00000668 4EB9000006CC             JSR      CSRINIT                GO START UP THE LANCE
 446    0000066E 08AC00050024   CHECK3    BCLR.B   #05,RINGSTAT(A4)       SEE IF THIS PORTION OF THE TEST IS COMPLETE
 447    00000674 67F8                     BEQ      CHECK3                 CHECK AGAIN
 448
 449                            ***   TURN OFF THE LANCE   ***
 450    00000676 247C00004002             MOVE.L   #RAP,A2                SELECT LANCE'S REGISTER ADDRESS PORT TO WRITE TO
 451    0000067C 34BC0000                 MOVE.W   #CSRO,(A2)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSRO
 452    00000680 267C00004000             MOVE.L   #RDP,A3                SELECT LANCE'S REGISTER DATA PORT TO WRITE TO
 453    00000686 36BC00FF                 MOVE.W   #$00FF,(A3)            TURN OFF THE LANCE
 454
 455                            ***   TEST EXTERNAL LOOPBACK   ***
 456
 457    0000068A 08AC00000024             BCLR.B   #00,RINGSTAT(A4)       CLEAR THE INTERNAL LOOP STATUS BIT
 458    00000690 00458300                 ORI.W    #$8300,D5              SET OWNERSHIP BIT IN XMIT RING
 459    00000694 3085                     MOVE.W   D5,(A0)                MOVE TMD1 TO RING LOCATION
```

```
Line S Location  Value          Source
 460   00000696 5488                     ADDQ.L    #02,A0                 GENERATE ADDRESS OF TMD2
 461   00000698 20BCFFE40000             MOVE.L    #$FFE40000,(A0)        MOVE BYTE COUNT(28 BYTES) TO TMD2 ON DESCRIPTOR
 462                            *                                         RING AND CLEAR OUT TMD3
 463   0000069E 32BC8004                 MOVE.W    #$8004,(A1)            SET UP INIT BLOCK TO INTERNAL LOOPBACK
 464   000006A2 4EB9000006CC             JSR       CSRINIT                GO START UP THE LANCE
 465   000006A8 08AC00050024    CHECK4   BCLR.B    #05,RINGSTAT(A4)       SEE IF THIS PORTION OF THE TEST IS COMPLETE
 466   000006AE 67F8                     BEQ       CHECK4                 CHECK AGAIN
 467
 468                            ***   TURN OFF THE LANCE   ***
 469   000006B0 247C00004002             MOVE.L    #RAP,A2                SELECT LANCE'S REGISTER ADDRESS PORT TO WRITE TO
 470   000006B6 34BC0000                 MOVE.W    #CSR0,(A2)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSR0
 471   000006BA 267C00004000             MOVE.L    #RDP,A3                SELECT LANCE'S REGISTER DATA PORT TO WRITE TO
 472   000006C0 36BC00FF                 MOVE.W    #$00FF,(A3)            TURN OFF THE LANCE
 473
 474   000006C4 08AC00040024             BCLR.B    #04,RINGSTAT(A4)       CLEAR DIAGNOSTIC BIT
 475   000006CA 4E75                     RTS
 476
 477
 478                            *******   END OF INITIALIZATION AND DIAGNOSTIC ROUTINE   ******
 479
 480                            *******************************************************
 481                            *                                                     *
 482                            *      INITIAL.DIAG.CSRINIT                           *
 483                            *      INITIALIZES THE CONTROL AND STATUS REGISTERS   *
 484                            *                                                     *
 485                            *******************************************************
 486
 487   000006CC 48E7FFFF        CSRINIT  MOVEM.L   A0-A7/D0-D7,-(A7)      SAVE OLD REGISTERS
 488   000006D0 207C00004002             MOVE.L    #RAP,A0                SELECT LANCE'S REGISTER ADDRESS PORT TO WRITE TO
 489   000006D6 30BC0001                 MOVE.W    #CSR1,(A0)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSR1 ADD.
 490   000006DA 227C00004000             MOVE.L    #RDP,A1                SELECT LANCE'S REGISTER DATA PORT TO WRITE TO
 491   000006E0 223C00002000             MOVE.L    #IADR,D1               GET INITIALIZATION BLOCK BASE ADDRESS
 492   000006E6 3281                     MOVE.W    D1,(A1)                LOAD INITIALIZATION BLOCK STARTING ADDRESS IN CSR1
 493   000006E8 30BC0002                 MOVE.W    #CSR2,(A0)             SELECT LANCE REGISTER ADDRESS PORT TO WRITE TO
 494   000006EC 4841                     SWAP      D1                     GET HIGH ORDER BYTE OF INITIALIZATION ADDRESS
 495   000006EE 3281                     MOVE.W    D1,(A1)                LOAD HIGH ORDER BYTE OF INIT. BLOCK INTO CSR2
 496   000006F0 30BC0003                 MOVE.W    #CSR3,(A0)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSR3
 497   000006F4 32BC0006                 MOVE.W    #$0006,(A1)            INITIALIZE LANCE TO INTERFACE TO MK68000 FORMAT
 498   000006F8 30BC0000                 MOVE.W    #CSR0,(A0)             LOAD REGISTER ADDRESS PORT OF LANCE WITH CSR0
 499   000006FC 32BC0043                 MOVE.W    #$0043,(A1)            INITIALIZES CSR0, LANCE IS NOW READY TO RUN
 500   00000700 4CDFFFFF                 MOVEM.L   (A7)+,D0-D7/A0-A7      RETRIVE OLD REGISTERS
 501   00000704 4E75                     RTS                              RETURN FROM THIS SUBROUTINE
 502
 503                            ***   END OF CSRINIT SUBROUTINE   ***
 504
 505
 506                            *******************************************************
 507                            *                                                     *
 508                            *      INITIAL.DIAG.CRC                               *
 509                            *      CYCLE REDUNDENCY CHECK SOFTWARE GENERATION     *
 510                            *                                                     *
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68

Line S Location  Value          Source
 511                            *                                                    *
 512                            *****************************************************
 513
 514    00000706 48E7FFFF       CRCGEN   MOVEM.L  A0-A7/D0-D7,-(A7)     SAVE OLD REGISTERS
 515    0000070A 207C00003094            MOVE.L   #$3094,A0             GET ADDRESS OF CRC LOCATION
 516    00000710 20BCB1109280            MOVE.L   #$B1109280,(A0)       MOVE CRC FOR AAA...AAA OUT TO BUFFER
 517    00000716 4CDFFFFF                MOVEM.L  (A7)+,D0-D7/A0-A7     RETRIVE OLD REGISTERS
 518    0000071A 4E75                    RTS                            RETURN FROM THIS SUBROUTINE
 519
 520                            ***   END OF CRCGEN SUBROUTINE   ***
 521
 522
 523    0000071C 004400040000   PROGIADR DC.W     MODE,PADR1,PADR2,PADR3,LADR1,LADR2,LADR3,LADR4,RDRA1,RDRA2,TDRA1,TDRA2
 524    00000734 0100           RBUFLEN  DC.W     BUFLEN
 525    00000736 0100           TBUFLEN  DC.W     BUFLEN
 526    00000738 0060           RBUFDISP DC.W     RBUFDIS
 527    0000073A 1020           TBUFDISP DC.W     TBUFDIS
 528
 529    0000073C 00000000                END

no errors detected.

Options in effect:

  NOA,BRL,NOCEX,CL,FRL,MC,MD,NOMEX,O,NOPCO,NOPCS,LIST,NOSTR,FORMAT,NOMOTOROLA
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68


SYMBOL TABLE LISTING


NAME        ATTR    SECT   VALUE


BUFLEN                   00000100
CALCDNE                  000004CA
CALCDONE                 0000049C
CHECK1                   00000602
CHECK2                   00000638
CHECK3                   0000066E
CHECK4                   000006A8
CLEAR                    00000400
COUNT16                  0000047C
COUNT20                  000004B2
CRCGEN                   00000706
CSR0        XDEF         00000000
CSR1        XDEF         00000001
CSR2        XDEF         00000002
CSR3        XDEF         00000003
CSRINIT                  000006CC
DATINTR     XREF    *    00000000
DIAG        XDEF         0000059A
DLENGTH     XDEF         00000034
FIRSTRCV                 000004FE
FIRSTXM                  0000053E
GOINIT                   00000590
IADR                     00002000
INCREM                   00000474
LADR1                    00000008
LADR2                    00000000
LADR3                    00000000
LADR4                    00000000
LANINTR     XREF    *    00000000
LASTRRNG    XDEF         00000014
LASTTRNG    XDEF         00000018
LOOPMCNT    XDEF         00000020
LOOPXADD    XDEF         00000028
MESS                     000005C6
MLENGTH     XDEF         00000038
MODE                     00000044
MORE        XDEF         0000002C
NEXTRRNG                 000004F4
NEXTTRNG    XDEF         0000001C
NXPOWER                  000004BE
NXTPOWER                 00000490
NXTTRNG                  00000534
ONE         XDEF         0000002E
PADR1                    00000004
PADR2                    00000000
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
INITIAL.A68


SYMBOL TABLE LISTING


NAME      ATTR   SECT   VALUE


PADR3                 00000000
PROGIADR              0000071C
RAP       XDEF        00004002
RBUFDIS               00000060
RBUFDISP              00000738
RBUFLEN               00000734
RDP       XDEF        00004000
RDRA1                 00002018
RDRA2                 00006000
RETURN                0000046A
RINGSTAT  XDEF        00000024
RINGSTK               000038C0
RLEN      XDEF        00000000
RRNGBASE  XDEF        00000004
RRNGBOT   XDEF        0000000C
STACK                 0000387C
TBUFDIS               00001020
TBUFDISP              0000073A
TBUFLEN   XDEF        00000736
TDRA1                 00002058
TDRA2                 00004000
TLEN      XDEF        00000002
TRINGCNT  XDEF        00000022
TRNGBASE  XDEF        00000008
TRNGBOT   XDEF        00000010
WAIT                  00000596
```

8.2 APPENDIX B, LANCE INTERRUPT ASSEMBLY CODE

The following pages provide a listing of the LANCE interrupt assembly code.

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68

Line S Location  Value          Source
   1 0
   2 0
   3 0
   4 0
   5 0                          *****************************************************************+
   6 0                          *+**************************************************************+*
   7 0                          **                                                              **
   8 0                          ** MODULE NAME: LEXCEPT                                         **
   9 0                          **                                                              **
  10 0                          ** AUTHOR:  JIM FONTAINE                                        **
  11 0                          **                                                              **
  12 0                          ** PROGRAM: LANCE                                               **
  13 0                          **                                                              **
  14 0                          ** LATEST REVISION DATE: JANUARY 20, 1984                       **
  15 0                          **                                                              **
  16 0                          ******************************************************************
  17 0                          **                                                              **
  18 0                          ** DESCRIPTION:  THIS MODULE IS THE LANCE INTERRUPT HANDLING    **
  19 0                          ** ROUTINE.  IT WILL EXAMINE THE CONTROL AND STATUS REGISTER    **
  20 0                          ** OF THE LANCE AND DETERMINE WHAT CAUSED LANCE TO INTERRUPT    **
  21 0                          ** THE HOST PROCESSOR.  AFTER IT DETERMINED WHAT CAUSED THE     **
  22 0                          ** THE INTERRUPT CAUSING CONDITION IT WILL SERVICE THE CON-     **
  23 0                          ** DITION OR REPORT THE ERROR.                                  **
  24 0                          **                                                              **
  25 0                          ****************************************************************+*
  26 0                          *****************************************************************+
  27 0
  28 0                          *************************************************
  29 0                          *                                               *
  30 0                          *               EQUATE TABLE                    *
  31 0                          *                                               *
  32 0                          *************************************************
  33 0
  34 0
  35 0
  36 0
  37 0          00003A20        OUTPUT   EQU      $3A20
  38 0
  39 0          00003B20        MESSAGE  EQU      $3B20
  40 0
  41 0          0000003C        TESTTMD1 EQU      $3C
  42 0
  43 0
  44 0
  45 0                                   XDEF     OUTPUT
  46 0                                   XDEF     MESSAGE
  47 0                                   XDEF     LANINTR
  48 0                                   XDEF     CRCERR
  49 0                                   XDEF     XRINGFIX
  50 0                                   XDEF     CLEANUP
  51 0                                   XDEF     PCKDONE
```

```
Line S Location  Value          Source
  52 0
  53 0                          *   KEYWORD VALUE REFERENCE   *
  54 0
  55 0                                   XREF      RAP
  56 0                                   XREF      RDP
  57 0
  58 0                                   XREF      CSR0
  59 0                                   XREF      CSR1
  60 0                                   XREF      CSR2
  61 0                                   XREF      CSR3
  62 0
  63 0                                   XREF      RLEN
  64 0                                   XREF      TLEN
  65 0                                   XREF      RRNGBASE
  66 0                                   XREF      TRNGBASE
  67 0                                   XREF      RRNGBOT
  68 0                                   XREF      TRNGBOT
  69 0                                   XREF      LASTRRNG
  70 0                                   XREF      LASTTRNG
  71 0                                   XREF      NEXTTRNG
  72 0                                   XREF      LOOPMCNT
  73 0                                   XREF      TRINGCNT
  74 0                                   XREF      RINGSTAT
  75 0                                   XREF      MLENGTH
  76 0                                   XREF      MORE
  77 0                                   XREF      ONE
  78 0                                   XREF      LOOPXADD
  79 0                                   XREF      DLENGTH
  80 0
  81 0                          *     SUBROUTINE CALLS REFERENCE    *
  82 0
  83 0                                   XREF      BABLSUB
  84 0                                   XREF      CERRSUB
  85 0                                   XREF      MISSSUB
  86 0                                   XREF      MERRSUB
  87 0                                   XREF      FRAMSUB
  88 0                                   XREF      CRCSUB
  89 0                                   XREF      OFLOSUB
  90 0                                   XREF      BUFFSUB
  91 0                                   XREF      RTRYSUB
  92 0                                   XREF      STOP
  93 0                                   XREF      LCARRSUB
  94 0                                   XREF      COLLSUB
  95 0                                   XREF      UFLOSUB
  96 0                                   XREF      BUFRSUB
  97 0                                   XREF      IDONSUB
  98 0
  99 0                                   XREF      SIZEERR
 100 0                                   XREF      XMITSERV
 101 0                                   XREF      PACKERR
 102 0                                   XREF      CRCBAD
```

UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68

```
Line S Location Value             Source
 103 0                                     XREF     LPACKERR
 104 0                                     XREF     LMESSBAD
 105 0
 106 0
 107 0
 108            00000800                   ORG      $800
 109
 110                              *******************************************************
 111                              *                                                     *
 112                              *   LANCE INTERRUPT ERROR CHECKING SUBMODULE          *
 113                              *   LEXCEPT.ERROR                                     *
 114                              *                                                     *
 115                              *******************************************************
 116
 117   00000800 397CCCCCFFFF      LANINTR  MOVE.W   #$CCCC,LOOPMCNT(A4)  TEST
 118   00000806 48E7FFFE                   MOVEM.L  D0-D7/A0-A6,-(A7)    SAVE REGS ON STACK
 119   0000080A 247CFFFFFFFF               MOVEA.L  #RAP,A2              SELECT LANCE'S RAP TO WRITE INTO
 120   00000810 34BCFFFF                   MOVE.W   #CSR0,(A2)           LOAD CSR0 ADDRESS IN REG ADD PORT
 121   00000814 267CFFFFFFFF               MOVEA.L  #RDP,A3              LOAD "REG DATA PORT" ADDRESS INTO REG A3
 122   0000081A 3413                       MOVE.W   (A3),D2              MOVE CONTENTS OF CSR0 INTO REG D2
 123   0000081C 3E02                       MOVE.W   D2,D7                GET A WORKING COPY OF CSR0
 124   0000081E 0247FFBF                   ANDI.W   #$FFBF,D7            MASK OUT INTERRUPT ENABLE BIT
 125   00000822 3687                       MOVE.W   D7,(A3)              DISABLE "INTR. ENABLE" BIT & CLEAR OUR THE REST
 126
 127                              **      DETERMINE WHAT CAUSED THE INTERRUPT   *
 128
 129   00000824 0802000F                   BTST     #$0F,D2              CHECK IF "ERROR" BIT IS SET IN CSR0
 130   00000828 672E                       BEQ.S    XMITCK               IF IT ISN'T, BRANCH TO XMIT INTR. CHECK
 131
 132   0000082A 0802000E          ERROR    BTST     #$0E,D2              CHECK IF IT IS A BABBLE ERROR
 133   0000082E 6706                       BEQ.S    CERR                 IF NOT, CHECK IF IT IS A COLLISION ERROR
 134   00000830 4EB9FFFFFFFF               JSR      BABLSUB              IF IT IS, JUMP TO BABBLE HANDLE ROUTINE
 135   00000836 0802000D          CERR     BTST     #$0D,D2              CHECK IF IT IS A COLLISION ERROR
 136   0000083A 6706                       BEQ.S    MISS                 IF NOT, CHECK IT IT IS A MISSED PACKET ERROR
 137   0000083C 4EB9FFFFFFFF               JSR      CERRSUB              IF IT IS, JUMP TO COLLISION HANDLE SUBR.
 138   00000842 0802000C          MISS     BTST     #$0C,D2              CHECK IF IT IS A MISSED PACKET ERROR
 139   00000846 4EB9FFFFFFFF               JSR      MISSSUB              IF IT IS, JUMP TO MISSED PACKET SUBR.
 140   0000084C 0802000B          MERR     BTST     #$0B,D2              CHECK IT IS A MEMORY ERROR
 141   00000850 6706                       BEQ.S    XMITCK               IF NOT, CHECK FOR XMIT INTR BIT SET
 142   00000852 4EB9FFFFFFFF               JSR      MERRSUB              IF IT IS, JUMP TO MEMORY ERROR SUBR.
 143
 144
 145                              ***********************************************
 146                              *                                             *
 147                              *     TRANSMIT INTERRUPT HANDLING ROUTINE     *
 148                              *     LEXCEPT.XMIT                            *
 149                              *                                             *
 150                              ***********************************************
 151
 152   00000858 08020009          XMITCK   BTST     #$09,D2              CHECK FOR "TRANSMIT INTR" BIT SET
 153   0000085C 670000BE                   BEQ      RVCRINT              IF IT ISN'T SEE IF RECEIVE INTERRUPT IS SET
```

```
Line S Location  Value         Source
 154
 155   00000860 206CFFFF                MOVE.L   LASTTRNG(A4),A0       LOAD UP NEXT XMIT RING ADDRESS
 156   00000864 2248                    MOVE.L   A0,A1                 GET A WORKING COPY OF THE RING ADDRESS
 157   00000866 5488                    ADD.L    #$02,A0               GENERATE ADDRESS OF TMD1 (RING STATUS)
 158   00000868 3010                    MOVE.W   (A0),D0               GET STATUS (TMD1) AND PUT IT IN REG D0
 159   0000086A 0800000C                BTST     #$0C,D0               CHECK FOR "MORE" BIT SET
 160   0000086E 6704                    BEQ.S    ONETEST               IF NOT, GO CHECK "ONE" BIT
 161   00000870 526CFFFF                ADD.W    #$01,MORE(A4)         IF SO, UPDATE "MORES" COUNTER
 162   00000874 0800000B       ONETEST  BTST     #$0B,D0               CHECK FOR "ONE" BIT SET
 163   00000878 6704                    BEQ.S    ERRORTST              IF NOT, GO CHECK "ERROR" BIT
 164   0000087A 526CFFFF                ADD.W    #$01,ONE(A4)          IF SO, UPDATA "ONE" COUNTER
 165
 166                           ******   TRANSMISSION ERROR DETERMINATION     *****
 167
 168   0000087E 0800000E       ERRORTST BTST     #$0E,D0               CHECK FOR "ERROR" BIT SET
 169   00000882 6764                    BEQ.S    LOOPTEST              IF NOT, CONTINUE WITH TRANSMIT ROUTINE
 170   00000884 5888                    ADD.L    #$04,A0               GENERATE ERROR STATUS ADDRESS
 171   00000886 3010                    MOVE.W   (A0),D0               GET ERROR STATUS AND PUT IT IN REG D0
 172   00000888 0800000A                BTST     #$0A,D0               CHECK FOR "RETRY" ERROR BIT SET
 173   0000088C 6712                    BEQ.S    LCARERR               IF NOT, CHECK FOR LOSS OF CARRIER
 174   0000088E 4EB9FFFFFFFF            JSR      STOP                  IF SO, STOP LANCE AND TAKE CARE OF ERROR
 175   00000894 3200                    MOVE.W   D0,D1                 MOVE TIME DOMAIN REFLECTOMETRY VALUE
 176   00000896 024103FF                ANDI.W   #$03FF,D1             MASK OFF STATUS BITS
 177   0000089A 4EB9FFFFFFFF            JSR      RTRYSUB               JUMP TO RETRY ERROR SUBROUTINE
 178   000008A0 0800000B       LCARERR  BTST     #$0B,D0               CHECK FOR "LOSS OF CARRIER" BIT SET
 179   000008A4 670C                    BEQ.S    LATECOLL              IF NOT, CHECK FOR LATE COLLISION
 180   000008A6 4EB9FFFFFFFF            JSR      STOP                  IF SO, STOP AND TAKE CARE OF IT
 181   000008AC 4EB9FFFFFFFF            JSR      LCARRSUB              JUMP TO "LOSS OF CARRIER" SUBROUTINE
 182   000008B2 0800000C       LATECOLL BTST     #$0C,D0               CHECK FOR "LATE COLLISION" BIT SET
 183   000008B6 670C                    BEQ.S    UFLOERR               IF NOT, CHECK FOR UNDER FLOW
 184   000008B8 4EB9FFFFFFFF            JSR      STOP                  IF SO, STOP AND TAKE CARE OF ERROR
 185   000008BE 4EB9FFFFFFFF            JSR      COLLSUB               JUMP TO "LATE COLLISION" SUBROUTINE
 186   000008C4 0800000E       UFLOERR  BTST     #$0E,D0               CHECK FOR "UNDERFLOW" BIT SET
 187   000008C8 670C                    BEQ.S    NOBUFF                IF NOT, CHECK FOR BUFFER ERROR
 188   000008CA 4EB9FFFFFFFF            JSR      STOP                  IF SO, STOP AND TAKE CARE OF IT
 189   000008D0 4EB9FFFFFFFF            JSR      UFLOSUB               JUMP TO "UNDERFLOW" SUBROUTINE
 190   000008D6 0800000F       NOBUFF   BTST     #$0F,D0               CHECK FOR "BUFFER ERROR" BIT SET
 191   000008DA 670C                    BEQ.S    LOOPTEST              IF NOT, CONTINUE WITH LOOPTEST
 192   000008DC 4EB9FFFFFFFF            JSR      STOP                  IF SO, STOP AND TAKE CARE OF ERROR
 193   000008E2 4EB9FFFFFFFF            JSR      BUFRSUB               JUMP TO "BUFFER ERROR" SUBROUTINE
 194
 195                           *******   LOOPBACK TESTING   ********
 196
 197   000008E8 082C0001FFFF   LOOPTEST BTST.B   #1,RINGSTAT(A4)       SEE IF WE ARE IN THE LOOP BACK MODE
 198   000008EE 6712                    BEQ.S    FINISH                IF NOT, FINISH UP WITH THE XMIT ROUTINE
 199   000008F0 2611                    MOVE.L   (A1),D3               IF IN LOOPBACK, GET XMIT BUFFER ADDRESS
 200   000008F2 E15B                    ROL.W    #8,D3                 POSITION THE UPPER BYTE OF LOWER WORD FOR CLEAR
 201   000008F4 163C0000                MOVE.B   #00,D3                CLEAR OUT STATUS PART OF RMD1
 202   000008F8 E05B                    ROR.W    #8,D3                 REPOSITION BYTES
 203   000008FA 4843                    SWAP     D3                    REFORMAT THE LONG WORD ADD FOR 68000 COMPATABIOLITY
 204   000008FC 2943FFFF                MOVE.L   D3,LOOPXADD(A4)       SAVE A COPY ON THE STACK FOR LATER
```

```
Line S Location  Value         Source
 205   00000900 601A                    BRA.S    RVCRINT                GO CHECK IF RECEIVE INTER. BIT IS SET
 206
 207                           ***   TRANSMIT RING RESTORATION    ***
 208
 209   00000902 066C0001FFFF   FINISH   ADDI.W   #$01,TRINGCNT(A4)      INCREMENT THE NUMBER OF AVAILABLE XMIT RINGS
 210   00000908 B3ECFFFF                CMPA.L   TRNGBOT(A4),A1         SEE IF WE ARE ABOUT TO WRAP AROUND
 211   0000090C 660A                    BNE.S    INCRE                  IF NOT, INCREMENT THE RING POINTER
 212   0000090E 296CFFFFFFFF            MOVE.L   TRNGBASE(A4),LASTTRNG(A4) MOVE THE RING BASE ADD ONTO LASTTRNG STACK SPOT
 213   00000914 60000192                BRA      IDONCHK                BRANCH TO CONT
 214   00000918 50ACFFFF       INCRE    ADD.L    #$08,LASTTRNG(A4)      UPDATE LAST RING ADDRESS
 215
 216                           **   END OF TRANSMIT INTERRUPT ROUTINE   *
 217
 218
 219                           ******************************************
 220                           *                                        *
 221                           *   RECEIVE INTERRUPT HANDLING ROUTINE   *
 222                           *   LEXCEPT.RVCR                         *
 223                           *                                        *
 224                           ******************************************
 225
 226   0000091C 0802000A       RVCRINT  BTST     #$0A,D2                CHECK FOR "RECEIVE INTR" BIT SET
 227   00000920 67000186                BEQ      IDONCHK                IF IT ISN'T, SEE IF INITIALIZATION DONE IS SET
 228
 229   00000924 206CFFFF       RVCRBUF  MOVE.L   LASTRRNG(A4),A0        GET DESCRIPTOR RING STARTING ADDR. (RMD0)
 230   00000928 2A48                    MOVE.L   A0,A5                  MAKE A WORKING COPY OF IT IN REG A5
 231   0000092A 5C8D                    ADDQ.L   #$06,A5                GENERATE ADDRESS OF RDM3
 232   0000092C 3215                    MOVE.W   (A5),D1                GET MESSAGE BYTE COUNT (MCNT)
 233   0000092E 9BFC00000004            SUBA.L   #04,A5                 GENERATE ADDRESS RMD1
 234
 235                           ***   ERROR CHECKING   ***
 236
 237   00000934 3015                    MOVE.W   (A5),D0                GET RECEIVED MESSAGE STATUS (RDM1)
 238   00000936 0800000E                BTST     #$0E,D0                SEE IF "ERROR" BIT IS SET
 239   0000093A 66000110                BNE      RECVERR                IF IS, FIND OUT WHAT CAUSED THE ERROR
 240
 241
 242                           ***   CHECK TO SEE IF WE ARE IN LOOPBACK MODE   ***
 243
 244   0000093E 082C0001FFFF            BTST.B   #1,RINGSTAT(A4)        CHECK TO SEE IF WE ARE IN LOOPBACK MODE
 245   00000944 670000B0                BEQ      NOLOOP                 IF WE ARE NOT IN LOOPBACK MODE CONTINUE
 246
 247                           ***   LOOPBACK MESSAGE SIZE CHECK   ***
 248
 249   00000948 B26CFFFF                CMP.W    MLENGTH(A4),D1         SEE IF RVCR MESSAGE IS EQUAL TO XMIT MESSAGE
 250   0000094C 67000008                BEQ      PACKCHK                IF SO CONTINUE
 251   00000950 4EB9FFFFFFFF            JSR      SIZEERR                IF NOT, SERVICE ERROR ROUTINE
 252
 253                           ***   LOOPBACK PACKET CHECK   ***
 254
 255   00000956 08000009       PACKCHK  BTST     #09,D0                 CHECK IF START OF PACKET BIT IS SET IN RMD1
```

```
Line S Location Value          Source
       0000095A 6606                    BNE.S    BITTST              IF IT IS, GO CHECK THE END OF PACKET BIT
       0000095C 4EB9FFFFFFFF            JSR      LPACKERR            IF IT IS NOT JUMP TO PACKET ERROR SUBR.
       00000962 08000008       BITTST   BTST     #08,D0              CHECK IF END OF PACKET BIT IS SET
       00000966 6606                    BNE.S    BADDGEN             IF IT IS, GO GENERATE BUFFER ADDRESS
       00000968 4EB9FFFFFFFF            JSR      LPACKERR            IF IT IS NOT, JUMP TO PACKET ERROR

                               ***   GET RVCR BUFFER STARTING ADDRESS   ***

       0000096E 2610           BADDGEN  MOVE.L   (A0),D3             MOVE RVCR BUFFER ADDRESS INTO REG D3
       00000970 E15B                    ROL.W    #08,D3              MOVE SECOND BYTE INTO POSITION TO BE CLEARED
       00000972 163C0000                MOVE.B   #$00,D3             CLEAR OUT THE STATUS
       00000976 E05B                    ROR.W    #$08,D3             REPOSITION BYTES
       00000978 4843                    SWAP     D3                  REFORMAT FOR 68000 COMPATIBILITY
       0000097A 3643                    MOVE     D3,A3               MOVE ADDRESS INTO AN ADDRESS REGISTER

                               ***   LOOPBACK XMIT-RVCR WORD COMPARISON   ***

       0000097C 322CFFFF                MOVE.W   DLENGTH(A4),D1      GET MESSAGE DATA LENGTH
       00000980 721C                    MOVE.L   #$01C,D1            LOAD D1 WITH 28 (28 BYTES IS THE DATA SIZE)
       00000982 226CFFFF                MOVE.L   LOOPXADD(A4),A1     GET LOOPBACK XMIT BUFFER STARTING ADDRESS
       00000986 3A13           MOREWDS  MOVE.W   (A3),D5             GET ONE WORD OF RVCR BUFFER MESSAGE
       00000988 3C11                    MOVE.W   (A1),D6             GET ONE WORD OF XMIT BUFFER MESSAGE
       0000098A BC45                    CMP.W    D5,D6               SEE IF THE TWO WORDS ARE IDENTICAL
       0000098C 6706                    BEQ.S    INCR                IF THEY ARE EQUAL, INCREMENT THE ADDRESS
       0000098E 4EB9FFFFFFFF            JSR      LMESSBAD            IF THEY ARE NOT EQUAL, JUMP TO ERROR MESSAGE
       00000994 548B           INCR     ADDQ.L   #$2,A3              OTHERWISE, INCREMENT ADDRESS
       00000996 5489                    ADDQ.L   #$2,A1              INCREMENT THE ADDRESS
       00000998 04410002                SUBI.W   #$02,D1             DECREMENT MESSAGE COUNT
       0000099C 6F02                    BLE.S    CRCCK               IF NO MORE WORDS, CHECK THE CRC
       0000099E 60E6                    BRA.S    MOREWDS             GO GET SOME MORE WORDS TO COMPARE

                               ***   SOFTWARE CRC TEST   ***

       000009A0 082C0004FFFF   CRCCK    BTST.B   #04,RINGSTAT(A4)    SEE IF WE ARE IN LANCE DIAGONOSTICS
       000009A6 6700001E                BEQ      XRINGFIX            IF NOT, GO FIX UP RINGS

       000009AA 7204                    MOVE.L   #$004,D1            LOAD D1 WITH 4 (4 BYTES IS THE CRC SIZE)
       000009AC 3A13           MORECRC  MOVE.W   (A3),D5             GET ONE WORD OF RVCR BUFFER MESSAGE
       000009AE 3C11                    MOVE.W   (A1),D6             GET ONE WORD OF XMIT BUFFER MESSAGE
       000009B0 BC45                    CMP.W    D5,D6               SEE IF THE TWO WORDS ARE IDENTICAL
       000009B2 6706                    BEQ.S    DECR                IF THEY ARE EQUAL, GO DECREMENT THE COUNT
       000009B4 4EB9FFFFFFFF            JSR      CRCBAD              IF THEY ARE NOT EQUAL, JUMP TO ERROR MESSAGE
       000009BA 04410002       DECR     SUBI.W   #02,D1              DECREMENT MESSAGE COUNT
       000009BE 6F06                    BLE.S    XRINGFIX            IF NO MORE WORDS, FIX UP THE XMIT RING
       000009C0 548B                    ADDQ.L   #2,A3               OTHERWISE, INCREMENT ADDRESS
       000009C2 5489                    ADDQ.L   #2,A1               INCREMENT THE ADDRESS
       000009C4 60E6                    BRA.S    MORECRC             GO GET SOME MORE WORDS TO COMPARE

                               ***   XMIT RING RESTORATION   ***

       000009C6 082C0004FFFF   XRINGFIX BTST.B   #04,RINGSTAT(A4)    SEE IF WE ARE IN LANCE DIAGONOSTICS
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68

Line S Location  Value          Source
 307   000009CC 660000CC                 BNE       OWNSHIP                 IF SO, LEAVE RINGS AS IS AND GIVE BACK OWNERSHIP
 308
 309   000009D0 066C0001FFFF    RINGFIX  ADDI.W    #$01,TRINGCNT(A4)       INCREMENT THE NUMBER A AVAILABLE XMIT RINGS
 310   000009D6 206CFFFF                 MOVE.L    LASTTRNG(A4),A0         GET LAST XMIT RING ADDRESS
 311   000009DA B1ECFFFF                 CMPA.L    TRNGBOT(A4),A0          SEE IF WE ARE DOWN TO THE BOTTOM OF THE RINGS
 312   000009DE 6A0C                     BPL.S     RINGGEN                 IF WE ARE, GO GENERATE NEW LAST RING
 313   000009E0 06AC00000008             ADDI.L    #$08,LASTTRNG(A4)       OTHERWISE, GENERATE NEW LAST RING VALUE, AND PUT ON STACK
                FFFF
 314   000009E8 60000096                 BRA       CLEANUP                 GO CLEAN UP RINGS
 315   000009EC 296CFFFFFFFF    RINGGEN  MOVE.L    TRNGBASE(A4),LASTTRNG(A4) MAKE RING BASE THE NEW LAST RING ADDRESS
 316   000009F2 6000008C                 BRA       CLEANUP                 GO CLEAN UP RINGS
 317
 318
 319
 320                            ********  "NON LOOPBACK" OPERATION **********
 321
 322                            **       PACKET DISCREPANCY CHECK    *
 323
 324   000009F6 08000009        NOLOOP   BTST      #09,D0                  IS THE START OF PACKET BIT SET IN RMD1?
 325   000009FA 6718                     BEQ.S     SETB4                   IF NOT, SEE IF IT WAS SET PREVIOUSLY
 326   000009FC 08AC0002FFFF             BCLR.B    #02,RINGSTAT(A4)        IF SO, SEE IF IT WAS SET PREVIOUSLY
 327
 328   00000A02 6600FFFF                 BNE       PACKERR                 IF IT WAS SET BEFORE YOU HAVE AN ERROR
 329   00000A06 08000008                 BTST      #08,D0                  IF NOT SET, CHECK IF END OF PACKET BIT IS SET
 330   00000A0A 661E                     BNE.S     PCKDONE                 IF IT IS, WE HAVE THE WHOLE MESSAGE,CHECK DONE
 331   00000A0C 002C0004FFFF             ORI.B     #$04,RINGSTAT(A4)       IF NOT, THIS IS ONLY THE START OF PACKET
 332   00000A12 6016                     BRA.S     PCKDONE                 GO ON, PACKET CHECKS OUT
 333   00000A14 08AC0002FFFF    SETB4    BCLR.B    #2,RINGSTAT(A4)         SEE IF START OF PACKET WAS SET PREVIOUSLY
 334   00000A1A 6700FFFF                 BEQ       PACKERR                 IF IT WAS NOT SET BEFORE YOU HAVE AN ERROR
 335   00000A1E 08000008                 BTST      #08,D0                  IF IT WAS SET, SEE IF END OF PACKET IS SET
 336   00000A22 6706                     BEQ.S     PCKDONE                 IF ENP SET, WE HAVE WHOLE MESSAGE, CHECK DONE
 337   00000A24 002C0004FFFF             ORI.B     #$04,RINGSTAT(A4)       IF NOT, THIS IS ONLY THE START OF MESSAGE
 338
 339                            **       CONTINUE WITH NORMAL PROCESS    *
 340
 341   00000A2A 2610            PCKDONE  MOVE.L    (A0),D3                 MOVE RVCR BUFFER ADDRESS INTO REG D3
 342   00000A2C E15B                     ROL.W     #08,D3                  MOVE SECOND BYTE INTO POSITION TO BE CLEARED
 343   00000A2E 163C0000                 MOVE.B    #$00,D3                 CLEAR OUT THE STATUS
 344   00000A32 E05B                     ROR.W     #$08,D3                 REPOSITION BYTES
 345   00000A34 4843                     SWAP      D3                      REFORMAT FOR 68000 COMPATIBILITY
 346   00000A36 2643                     MOVE.L    D3,A3                   MOVE ADDRESS INTO AN ADDRESS REGISTER
 347
 348                            **       MOVE RECEIVED MESSAGE FROM RECEIVE BUFFER INTO OUTPUT DEVICE    *
 349
 350   00000A38 227C00003A20             MOVE.L    #OUTPUT,A1              MOVE OUTPUT ADDRESS INTO REG A1
 351   00000A3E 3293            SOMEMORE MOVE.W    (A3),(A1)               MOVE MESSAGE WORD TO OUTPUT
 352   00000A40 548B                     ADD.L     #$2,A3                  INCREMENT THE MESSAGE BUFFER ADDRESS
 353   00000A42 5489                     ADD.L     #$2,A1                  INCREMENT THE MESSAGE OUTPUT ADDRESS
 354   00000A44 04410002                 SUBI.W    #02,D1                  DECREMENT THE MESSAGE LENGTH REGISTER
 355   00000A48 6EF4                     BGT.S     SOMEMORE                IF THERE ARE SOME MORE WORDS TO XFER, DO IT
 356   00000A4A 6034                     BRA.S     CLEANUP                 FINISHED, GO CLEANUP RINGS
```

```
Line S Location  Value         Source

                               ********  RECEIVE MESSAGE ERROR DETERMINATION  *******

       00000A4C 4EB9FFFFFFFF   RECVERR  JSR      STOP                     STOP AND FIND OUT WHAT THE ERROR IS
       00000A52 0800000D                BTST     #$0D,D0                  CHECK IF "FRAME ERROR" BIT IS SET
       00000A56 6712                    BEQ.S    OFLOERR                  IF NOT, CHECK IF OVER FLOW ERROR SET
       00000A58 4EB9FFFFFFFF            JSR      FRAMSUB                  IF SO, JUMP TO FRAM ERROR SUBROUTINE
       00000A5E 0800000B       CRCERR   BTST     #$0B,D0                  CHECK IF "CRC ERROR" BIT IS SET
       00000A62 6706                    BEQ.S    OFLOERR                  IF NOT, CHECK OVERFLOW ERROR
       00000A64 4EB9FFFFFFFF            JSR      CRCSUB                   IF SO JUMP TO CRC ERROR SUBROUTINE
       00000A6A 0800000C       OFLOERR  BTST     #$0C,D0                  CHECK IF "OVER FLOW ERROR" BIT IS SET
       00000A6E 6706                    BEQ.S    BUFF                     IF NOT, CHECK BUFFER ERROR
       00000A70 4EB9FFFFFFFF            JSR      OFLOSUB                  IF SO JUMP TO OVER FLOW SUBROUTINE
       00000A76 0800000A       BUFF     BTST     #$0A,D0                  CHECK IF "BUFFER ERROR" BIT IS SET
       00000A7A 4EB9FFFFFFFF            JSR      BUFFSUB                  IF SO, JUMP TO BUFFER ERROR SUBROUTINE

                               ******   CLEANUP     ******

       00000A80 226CFFFF       CLEANUP  MOVE.L   LASTRRNG(A4),A1          GO GET LAST RECEIVE RING ADDRESS
       00000A84 B3ECFFFF                CMPA.L   RRNGBOT(A4),A1           SEE IF WE ARE DOWN TO THE BOTTOM OF THE RINGS
       00000A88 6A0A                    BPL.S    RINGLAST                 IF WE ARE, GO GENERATE NEW LAST RING
       00000A8A 06AC00000008            ADDI.L   #$08,LASTRRNG(A4)        GENERATE NEW LAST RING VALUE & PUT ON STACK
                FFFF
       00000A92 6006                    BRA.S    OWNSHIP                  GO GIVE OWNERSHIP BACK TO LANCE
       00000A94 296CFFFFFFFF   RINGLAST MOVE.L   RRNGBASE(A4),LASTRRNG(A4) MAKE RING BASE THE NEW LAST RING ADDRESS
       00000A9A 1ABC0080       OWNSHIP  MOVE.B   #$80,(A5)                GIVE OWNERSHIP BACK TO LANCE

       00000A9E 082C0002FFFF            BTST.B   #02,RINGSTAT(A4)         SEE IF START OF PACKET BIT IS SET
       00000AA4 6600FE7E                BNE      RVCRBUF                  IF SO, THERE IS MORE THAT ONE BUFFER


                               **      END OF RVCR INTERRUPT ROUTINE *

                               *******************************************
                               *                                         *
                               *    LANCE INITIALIZATION DONE INTERRUPT  *
                               *    LEXCEPT.IDON                         *
                               *                                         *
                               *******************************************

       00000AA8 08020008       IDONCHK  BTST     #$08,D2                  CHECK FOR "INITIALIZATION DONE" BIT SET
       00000AAC 6716                    BEQ.S    RESTORE                  IF NOT SET, RESTORE AND FINISH UP
       00000AAE 4EB9FFFFFFFF            JSR      IDONSUB                  IF IT IS, JUMP TO INITIALIZATION DONE SUBR.

       00000AB4 082C0004FFFF            BTST.B   #04,RINGSTAT(A4)         SEE IF WE ARE IN LANCE DIAGONOSTICS
       00000ABA 67000008                BEQ      RESTORE                  IF NOT CONTIUNE NORMALLY
       00000ABE 08EC0005FFFF            BSET.B   #5,RINGSTAT(A4)          IF SO, SET "TEST FINISHED" BIT
                               *
       00000AC4 367CFFFF       RESTORE  MOVE.W   #RDP,A3                  MOVE ADDRESS OF REGISTER DATA PORT TO A3
       00000AC8 36BC0040                MOVE.W   #$0040,(A3)              SET THE INTERRUPT ENABLE BIT IN CSR0
                               *
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68

Line S Location  Value         Source
 407   00000ACC 4CDF7FFF               MOVEM.L  (A7)+,A0-A6/D0-D7     RETRIEVE OLD REGISTERS
 408   00000AD0 4E73                   RTE                            RETURN FROM EXCEPTION HANDLING
 409   00000AD2 00000000               END

no errors detected.

Options in effect:

  NOA,BRL,NOCEX,CL,FRL,MC,MD,NOMEX,O,NOPCO,NOPCS,LIST,NOSTR,FORMAT,NOMOTOROLA
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68


SYMBOL TABLE LISTING


NAME      ATTR   SECT   VALUE


BABLSUB   XREF    *   00000000
BADDGEN               0000096E
BITTST                00000962
BUFF                  00000A76
BUFFSUB   XREF    *   00000000
BUFRSUB   XREF    *   00000000
CERR                  00000836
CERRSUB   XREF    *   00000000
CLEANUP   XDEF        00000A80
COLLSUB   XREF    *   00000000
CRCBAD    XREF    *   00000000
CRCCK                 000009A0
CRCERR    XDEF        00000A5E
CRCSUB    XREF    *   00000000
CSR0      XREF    *   00000000
CSR1      XREF    *   00000000
CSR2      XREF    *   00000000
CSR3      XREF    *   00000000
DECR                  000009BA
DLENGTH   XREF    *   00000000
ERROR                 0000082A
ERRORTST              0000087E
FINISH                00000902
FRAMSUB   XREF    *   00000000
IDONCHK               00000AA8
IDONSUB   XREF    *   00000000
INCR                  00000994
INCRE                 00000918
LANINTR   XDEF        00000800
LASTRRNG  XREF    *   00000000
LASTTRNG  XREF    *   00000000
LATECOLL              000008B2
LCARERR               000008A0
LCARRSUB  XREF    *   00000000
LMESSBAD  XREF    *   00000000
LOOPMCNT  XREF    *   00000000
LOOPTEST              000008E8
LOOPXADD  XREF    *   00000000
LPACKERR  XREF    *   00000000
MERR                  0000084C
MERRSUB   XREF    *   00000000
MESSAGE   XDEF        00003B20
MISS                  00000842
MISSSUB   XREF    *   00000000
MLENGTH   XREF    *   00000000
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
LEXCEPT.A68


SYMBOL TABLE LISTING


NAME       ATTR    SECT   VALUE


MORE       XREF     *    00000000
MORECRC                  000009AC
MOREWDS                  00000986
NEXTTRNG   XREF     *    00000000
NOBUFF                   000008D6
NOLOOP                   000009F6
OFLOERR                  00000A6A
OFLOSUB    XREF     *    00000000
ONE        XREF     *    00000000
ONETEST                  00000874
OUTPUT     XDEF          00003A20
OWNSHIP                  00000A9A
PACKCHK                  00000956
PACKERR    XREF     *    00000000
PCKDONE    XDEF          00000A2A
RAP        XREF     *    00000000
RDP        XREF     *    00000000
RECVERR                  00000A4C
RESTORE                  00000AC4
RINGFIX                  000009D0
RINGGEN                  000009EC
RINGLAST                 00000A94
RINGSTAT   XREF     *    00000000
RLEN       XREF     *    00000000
RRNGBASE   XREF     *    00000000
RRNGBOT    XREF     *    00000000
RTRYSUB    XREF     *    00000000
RVCRBUF                  00000924
RVCRINT                  0000091C
SETB4                    00000A14
SIZEERR    XREF     *    00000000
SOMEMORE                 00000A3E
STOP       XREF     *    00000000
TESTTMD1                 0000003C
TLEN       XREF     *    00000000
TRINGCNT   XREF     *    00000000
TRNGBASE   XREF     *    00000000
TRNGBOT    XREF     *    00000000
UFLOERR                  000008C4
UFLOSUB    XREF     *    00000000
XMITCK                   00000858
XMITSERV   XREF     *    00000000
XRINGFIX   XDEF          000009C6
```

8.3 APPENDIX C, MESSAGE INTERRUPT ASSEMBLY CODE

The following pages provide a listing of the message interrupt assembly code.

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
MESSAGE.A68

Line S Location  Value          Source
   1 0                          *******************************************************+
   2 0                          *+****************************************************+*
   3 0                          **                                                    **
   4 0                          ** MODULE NAME: MESSAGE                               **
   5 0                          **                                                    **
   6 0                          ** AUTHOR:  JIM FONTAINE                              **
   7 0                          **                                                    **
   8 0                          ** PROGRAM: LANCE                                     **
   9 0                          **                                                    **
  10 0                          ** LATEST REVISION DATE: JANUARY 20, 1958             **
  11 0                          **                                                    **
  12 0                          ********************************************************
  13 0                          ** DESCRIPTION:  THIS MODULE GENERATES A PSEUDO       **
  14 0                          ** MESSAGE THAT IS THEN PUT INTO THE TRANSMIT BUFFER. **
  15 0                          ** THE MESSAGE CODE AND LENGTH IS DETERMINED BY THE   **
  16 0                          ** DLENGTH FOUND IN MEMORY LOCATION DLENGTH(A4)       **
  17 0                          ** THE BUFFER IS FILLED WITH CONSECUTIVE COUNTS,      **
  18 0                          ** NUMBERING FROM $0000 TO $(DLENGTH - 1). DLENGTH    **
  19 0                          ** MUST BE GREATER THAT 63 FOR NORNAL DATA TRANS-     **
  20 0                          ** MISSION AND LESS THAN 33 FOR LOOPBACK TRANS-       **
  21 0                          ** MISSION.                                           **
  22 0                          ******************************************************+*
  23 0                          *******************************************************+
  24 0
  25 0
  26 0                          *************************************************
  27 0                          *                                               *
  28 0                          *               EQUATE TABLE                    *
  29 0                          *                                               *
  30 0                          *************************************************
  31 0
  32 0
  33 0
  34 0                          *   DESCRIPTOR RING STACK ALLOCATION   *
  35 0
  36 0
  37 0                          *   KEYWORD VALUE DEFINITION   *
  38 0
  39 0
  40 0                                   XREF     RLEN
  41 0                                   XREF     TLEN
  42 0                                   XREF     TRNGBASE
  43 0                                   XREF     TRNGBOT
  44 0                                   XREF     LASTTRNG
  45 0                                   XREF     NEXTTRNG
  46 0                                   XREF     LOOPMCNT
  47 0                                   XREF     TRINGCNT
  48 0                                   XREF     RINGSTAT
  49 0                                   XREF     LOOPXADD
  50 0                                   XREF     MLENGTH
  51 0                                   XREF     MESSAGE
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
MESSAGE.A68


SYMBOL TABLE LISTING


NAME       ATTR   SECT   VALUE


BOTHBITS                00000B74
DATINTR    XDEF         00000B00
DLENGTH    XREF     *   00000000
ENDBIT                  00000B6E
ENDPAC                  00000B66
LASTTRNG   XREF     *   00000000
LOOPMCNT   XREF     *   00000000
LOOPXADD   XREF     *   00000000
MEMORY                  00003FF0
MESSAGE    XREF     *   00000000
MLENGTH    XREF     *   00000000
MOREDATA                00000B48
NEXTRNG                 00000B12
NEXTTRNG   XREF     *   00000000
NORINGS    XREF     *   00000000
NOUP                    00000B3C
OUTPUT     XREF     *   00000000
RINGSTAT   XREF     *   00000000
RLEN       XREF     *   00000000
STARTPAC                00000B7A
STOP       XREF     *   00000000
TBUFLEN    XREF     *   00000000
TLEN       XREF     *   00000000
TMDFIX                  00000B8C
TRINGCNT   XREF     *   00000000
TRNGBASE   XREF     *   00000000
TRNGBOT    XREF     *   00000000
UPDATE                  00000B34
XMITDONE   XDEF         00000BB8
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
MESSAGE.A68

Line S Location  Value          Source
  52 0                                   XREF     OUTPUT
  53 0                                   XREF     DLENGTH
  54 0                                   XREF     TBUFLEN
  55 0
  56 0                          *     SUBROUTINE CALLS REFERENCE    *
  57 0
  58 0                                   XREF     STOP
  59 0                                   XREF     NORINGS
  60 0
  61 0                                   XDEF     XMITDONE
  62 0                                   XDEF     DATINTR
  63 0
  64 0          00003FF0        MEMORY   EQU      $3FF0
  65 0
  66 0
  67 0                          ***  MAIN PROGRAM   ***
  68 0                          *
  69 0
  70            00000B00                 ORG      $B00
  71
  72
  73   00000B00 48E7FFFF        DATINTR  MOVEM.L  A0-A7/D0-D7,-(A7)     SAVE OLD REGISTERS
  74   00000B04 4280                     CLR.L    D0
  75   00000B06 4281                     CLR.L    D1                    CLEAR OUT REG
  76   00000B08 4282                     CLR.L    D2
  77   00000B0A 4284                     CLR.L    D4
  78   00000B0C 4285                     CLR.L    D5
  79
  80   00000B0E 302CFFFF                 MOVE.W   DLENGTH(A4),D0        GET MESSAGE DATA SIZE
  81   00000B12 046C0000FFFF    NEXTRNG  SUBI.W   #00,TRINGCNT(A4)      SEE IF THERE ARE TRANSMIT RINGS AVAILABLE
  82   00000B18 6700FFFF                 BEQ      NORINGS               IF NONE LEFT, GIVE ERROR
  83   00000B1C 046C0001FFFF             SUBI.W   #$01,TRINGCNT(A4)     IF THERE ARE SOME, DECREMENT THE COUNT
  84   00000B22 206CFFFF                 MOVE.L   NEXTTRNG(A4),A0       GET XMIT RING BASE ADDRESS
  85
  86   00000B26 B1ECFFFF                 CMPA.L   TRNGBOT(A4),A0        SEE IF WE ARE ABOUT TO WRAP AROUND
  87   00000B2A 6608                     BNE.S    UPDATE                IF NOT UPDATE AS USUAL
  88   00000B2C 296CFFFFFFFF             MOVE.L   TRNGBASE(A4),NEXTTRNG(A4) IF SO, POINT TO TOP OF RING STACK
  89   00000B32 6008                     BRA.S    NOUP                  BRANCH TO NO UPDATE
  90
  91   00000B34 06AC00000008    UPDATE   ADDI.L   #$08,NEXTTRNG(A4)     UPDATE THE ADDRESS OF THE NEXT XMIT RING
                FFFF
  92   00000B3C 2610            NOUP     MOVE.L   (A0),D3               GET XMIT BUFFER BASE ADD
  93   00000B3E 024300FF                 ANDI.W   #$00FF,D3             CLEAR STATUS OUT OF REGISTER
  94   00000B42 3A03                     MOVE.W   D3,D5                 STORE THE TMD1 WORD IN REG D5 FOR UPDATE
  95   00000B44 4843                     SWAP     D3                    MOVE WORDS FOR 68000 COMPATIBILITY
  96   00000B46 2643                     MOVE.L   D3,A3                 MOVE XMIT BUFFER ADD. INTO AN ADD. REG
  97   00000B48 068100000001    MOREDATA ADDI.L   #$01,D1               GENERATE NEXT MESSAGE BYTE TO OUTPUT
  98   00000B4E 1681                     MOVE.B   D1,(A3)               MOVE DATA BYTE INTO OUTPUT MEMORY LOCATION
  99   00000B50 06420001                 ADDI.W   #$01,D2               INCREMENT BYTE COUNT 1 (TOTAL WORD COUNT)
 100   00000B54 06440001                 ADDI.W   #$01,D4               INCREMENT BYTE COUNT 2 (BUFFER WORD COUNT)
 101   00000B58 B042                     CMP.W    D2,D0                 SEE IF THERE ARE MORE DATA WORDS TO OUTPUT
```

```
UNITED TECHNOLOGIES MOSTEK 68000 Assembler V1.4
Site 99994 MOSTEK
MESSAGE.A68

Line S Location Value            Source
 102    00000B5A 670A                     BEQ.S    ENDPAC                IF NOT, THIS IS THE END OF THE PACKET
 103    00000B5C 0C44FFFF                 CMP.W    #TBUFLEN,D4           IF SO, SEE IF THERE IS MORE BUF SPACE LEFT
 104    00000B60 6718                     BEQ.S    STARTPAC              IF NOT, GO CHECK IF THIS IS THE START OF PAC
 105    00000B62 528B                     ADDQ.L   #$01,A3               IF SO, INCREMENT BUFFER ADD & GET NEXT WORD
 106    00000B64 60E2                     BRA.S    MOREDATA              GO GET ANOTHER WORD TO PUT INTO THE BUFFER
 107
 108                             * CHECK IF START & END BITS ARE SET  *
 109
 110    00000B66 082C0002FFFF    ENDPAC   BTST.B   #02,RINGSTAT(A4)      SEE IF START BIT HAD BEEN SET BEFORE
 111    00000B6C 6706                     BEQ.S    BOTHBITS              IF NOT, SET BOTH START & END OF PAC BITS IN TMD1
 112    00000B6E 00450100        ENDBIT   ORI.W    #$0100,D5             OTHERWISE SET ONLY THE END OF PACKET BIT
 113    00000B72 6018                     BRA.S    TMDFIX                GO FINISH UP TRANSMIT DESCRIPTOR 1
 114    00000B74 00450200        BOTHBITS ORI.W    #$0200,D5             SET END OF PACKET BIT IN TMD1
 115    00000B78 60F4                     BRA.S    ENDBIT                NOT GO SET END OF PACKET BIT
 116    00000B7A 082C0002FFFF    STARTPAC BTST.B   #02,RINGSTAT(A4)      SEE IF START BIT HAD BEEN SET BEFORE
 117    00000B80 660A                     BNE.S    TMDFIX                IF SO GO FINISH OF TMD1 STATUS
 118    00000B82 00450200                 ORI.W    #$0200,D5             IF NOT SET START OF PACKET BIT IN TMD1
 119    00000B86 08EC0002FFFF             BSET.B   #02,RINGSTAT(A4)      SET THE START STATUS BIT IN THE RINGSTATUS
 120
 121                             *  FIX UP THE TRANSMIT DESCRIPTOR RINGS  *
 122
 123    00000B8C 5888            TMDFIX   ADDQ.L   #04,A0                GENERATE ADDRESS OF TMD2
 124    00000B8E 0A44FFFF                 EORI.W   #$FFFF,D4             TAKE 2'S COMPLEMENT OF WORD COUNT
 125    00000B92 06440001                 ADDI.W   #$01,D4
 126    00000B96 0044F000                 ORI.W    #$F000,D4             FORCE FOUR MSB TO ONES
 127    00000B9A 3084                     MOVE.W   D4,(A0)               MOVE MESSAGE BYTE COUNT INTO TMD2
 128    00000B9C 4284                     CLR.L    D4                    CLEAR OUT MESSAGE COUNT FOR NEXT BUFFER
 129    00000B9E 5488                     ADDQ.L   #$02,A0               GENERATE ADDRESS OF TMD3
 130    00000BA0 30BC0000                 MOVE.W   #$0000,(A0)           CLEAR OUT TMD3
 131    00000BA4 5988                     SUBQ.L   #04,A0                GENERATE ADDRESS OF TMD1
 132    00000BA6 00458000                 ORI.W    #$8000,D5             SET OWN BIT, GIVING OWNERSHIP TO LANCE
 133    00000BAA 3085                     MOVE.W   D5,(A0)               PUT TMD1 BACK INTO XMIT RING
 134
 135                             *  CHECK TO SEE IF WE NEED TO GENERATE ANOTHER MESSAGE BUFFER   *
 136
 137    00000BAC B042                     CMP.W    D2,D0                 SEE IF THERE ARE MORE DATA WORDS TO OUTPUT
 138    00000BAE 6600FF62                 BNE      NEXTRNG               IF THERE ARE, SEE IF ANY BUFFER RINGS LEFT
 139    00000BB2 08AC0002FFFF             BCLR.B   #02,RINGSTAT(A4)      CLEAR OUT THE START OF PACKET BIT ON RINGSTAT
 140
 141    00000BB8 4CDFFFFF        XMITDONE MOVEM.L  (A7)+,D0-D7/A0-A7     RETRIVE OLD REGISTERS
 142
 143    00000BBC 4E73                     RTE                            RETURN FROM EXCEPTION PROCESSION
 144    00000BBE 00000000                 END

no errors detected.

Options in effect:

  NOA,BRL,NOCEX,CL,FRL,MC,MD,NOMEX,O,NOPCO,NOPCS,LIST,NOSTR,FORMAT,NOMOTOROLA
```